A PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

III, NÚMERO 40

975 Ptas.

DIRECT X
Programación de gráficos con Delphi

PERT
Evaluación de procesos

OBJETOS EN VISUAL
Clases, objetos y herencia con VB 5.0

LINUX
Conexión a Internet mediante Linux

JUEGOS
Generador de movimientos de ajedrez

CONTENIDO DEL CD-ROM

- XFree86 3.3.1
- X Appeal
- 3D WinBench
- Clientes CICS
- Internet Explorer 4.0 para Windows 95
- Aplicaciones de programación JAVA
- Y mucho más

SISTEMAS DISTRIBUIDOS

8 "413042"303299"

SÓLO PROGRAMADORES

Número 40
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Director Adjunto
Fernando Escudero
fescuder@towercom.es
Colaboradores
E. de la Lastra, Eugenio Castillo, Ernesto Schmitz, Alejandro Reyero, Chema Álvarez, J. Antonio Mendoza, Jorge Delgado, Bonifacio Villalobos, Francisco Goyá, Antonio J. Novillo, Constantino Sánchez, Jordi Agost.
Maquetación
Susana Llano
Tratamiento de Imagen
María Arce Giménez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Isabel Bojo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Óscar Rodríguez (jefe)
Javier Amado
Servicio Técnico
Óscar Casado

Filmación
Megatipo
Impresión
G. Don Bosco
Distribución
SGEL
Distribución en Argentina
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN

EDITORIAL

¡Más madera, es la guerra!

Quién nos iba a decir a nosotros que el absurdo y genial humor de los Hermanos Marx, que reducían a astillas su propio tren para poder seguir viajando en él, se vería reflejado, absurdamente, en el muy pocas veces genial mundo de la informática. Con el avance constante y espectacular del hardware, los programadores se han relajado y han preferido, antes que mejorar la calidad de su trabajo mediante una introspección fecunda, subirse sin rodeos a cada nuevo tren que pasa en una extraversión estéril. Olvidándose de la perfecta sencillez del buen hacer, han empezado a producir verdaderos monstruos en serie, para los cuales, claro es, cada vez hace falta un hardware más potente...

A la furibunda orden de Groucho: "¡Más madera, es la guerra!" se han sumado tanto los productores de hardware como los desarrolladores de software, formando una clásica pescadilla que se muerde la cola, una *variatio* del eterno retorno tan frenética que hasta al propio Nietzsche le daría vértigo.

Quizá cuando alguien se pregunte por el sentido que tiene un avance alocado sin la mesura y el cuidado que produce una meta, cualquiera que sea, lleguemos a algún remanso de tranquilidad que nos permita contemplar lo que hemos creado, arrojar por la borda todo el lastre inútil que, paradójicamente, nos impide avanzar y nos empuja hacia adelante y, en un momento de clarividencia, separar, en definitiva, la paja del heno y sustentar el avance sobre sólidas bases.

Sin embargo, es algo que tiene que surgir del mismo centro de este interminable círculo vicioso, pues un simple observador no puede hacer otra cosa que elevar su voz. Para que un movimiento crítico de la programación de este cariz tuviera cierta relevancia debería nacer en las raíces mismas del problema. Pero, ¿acaso hay alguien verdaderamente dispuesto a llevarlo a cabo?

De todas maneras, por el momento lo único que nos compete a nosotros es la programación. El tema central de este número gira alrededor de los sistemas distribuidos, que ponen su granito de arena a esta polémica aprovechando de forma más efectiva los recursos existentes en todo tipo de redes.

Tenéis a vuestra disposición la dirección de correo electrónico de la revista: solop@towercom.es, donde podéis enviar vuestras sugerencias sobre los contenidos, los temas que queráis que incluyamos y los productos que os gustaría que distribuyéramos con el CD-ROM, que todos los meses regalamos. Sois nuestros mejores colaboradores.

SÓLO PROGRAMADORES





LA REVITALICACIÓN DEL CÓDIGO COBOL

NetExpress de Micro Focus, la solución para crear, mantener y desplegar aplicaciones de Internet, incorporando la lógica de negocio de la empresa

El entorno NetExpress ofrece un conjunto de herramientas de desarrollo que incluye asistentes de diseño. Estas reglas pre-programadas y las plantillas generadoras de código permiten a los programadores desentenderse de las complejidades de los entornos de la Web y centrarse en los procesos de negocio, en vez de hacerlo en las tecnologías de Internet subyacentes. NetExpress también incluye potentes capacidades de debugging y evaluación, lo que permite a los programadores examinar y cambiar rápidamente cualquier dato en el programa durante la ejecución. De esta manera, se reduce considerablemente el ciclo de desarrollo total.

NetExpress se construye en los puntos fuertes tradicionales de COBOL, añadiendo una gama de funcionalidades modernas. Se incluye el COBOL Dialect Support, que facilita la migración de la lógica de negocio existente en COBOL hacia el entorno NetExpress y proporciona la posibilidad de portar fácilmente aplicaciones a un extenso rango de entornos operativos y plataformas hardware.

Micro Focus ha anunciado el lanzamiento de NetExpress, la primera solución que permite a las grandes compañías reutilizar el código COBOL para proporcionar acceso a datos corporativos críticos, utilizando las tecnologías Web.

NetExpress ofrece el primer entorno de desarrollo completo e integrado (IDE) para crear y desplegar aplicaciones de Internet e Intranet, incorporando la lógica de negocio de la empresa. NetExpress permite a las compañías dar respuesta a las necesidades cambiantes de sus clientes, proveedores y empleados posibilitando el acceso a la información crítica a través de Internet e Intranets.

Con un 70% de la información de negocio mundial residiendo en *mainframes* y con 150.000 millones de líneas de COBOL existentes hoy en día, los negocios necesitan una manera efectiva de reutilizar sus *mainframes* heredados y sus inversiones en lógica de negocio. Esto ha creado un mercado emergente para empresas que ofrecen soluciones para la integración de la informática corporativa y la Web, que se estima alcanzará los 15.000 millones de pesetas para el año 2000, frente a los 4.900 millones de 1996 según IDC.

SUN E IBM CERTIFICAN EL BUEN ESTADO DE SALUD DE UNIX

Unix no está muerto

Unix está lejos de estar herido de muerte como afirman los partidarios más radicales del NT Microsoft. De hecho, si tenemos en cuenta lo que afirman tanto la empresa Sun Microsystem como la IBM, goza de una excelente salud. Ambos fabricantes han anunciado importantes avances en sus estaciones de trabajo que están basadas en Unix. Sun Mocrosystem ha revelado detalles de un chip que posibilitará que sus UltraSPARC de 64 bits pasen sin problemas la barrera de los 600 MHz. a mediados del próximo año. Esta nueva generación de UltraSPARC, denominada UltraSPARC III, podría alcanzar hasta los 800 MHz. en posteriores desarrollos.

IBM anunció el lanzamiento de su primer hard/software de 64 bits, formado por el servidor RS/6000 S70 y la versión 4.3 del sistema operativo AIX.

Esto muestra que el lanzamiento de nuevos sistemas basados, con nuevas prestaciones, en Unix es continuo, y no se ha visto afectado por la aparición de la plataforma Windows NT.

No obstante, Sun ha anunciado que seguirá sin implementar Windows NT, ya que considera que el lenguaje Unix presenta sobre NT múltiples ventajas. IBM, sin embargo, sí vende servidores que corren bajo Windows NT.

Computer Associates y Apple integran Jasmine y Yellow Box en una solución multimedia orientada a objetos

Computer Associates International y Apple Computer han anunciado la integración de la herramienta de bases de datos orientadas a objeto, Jasmine, y el entorno de desarrollo de aplicaciones Yellow Box, componente clave de la próxima generación OS de Apple denominada Rhapsody.

Jasmine de CA combina una base de datos orientada a objetos con un sistema de desarrollo multimedia. Soporta todas las funcionalidades que ofrecen a las bases de datos orientadas a objeto su potencia, entra las que cabe citar legado, propiedades a nivel de ejemplo y clase, métodos de ejemplo y clase. Las librerías de Jasmine gestionan datos multimedia, incluyendo gráficos, animación, sonido y vídeo. Una librería de clase SQL permite a Jasmine acceder a los datos heredados actualizados.

Los desarrolladores de Yellow Box se beneficiarán de las funcionalidades sencillas de orientación a objetos y bases de Jasmine, junto con las características escalables y de implementación de misión crítica. El soporte nativo de objetos multimedia de Jasmine, tales como vídeo, sonido, imágenes y texto, la convierte en la plataforma ideal para las aplicaciones actuales.

Los desarrolladores de Jasmine aprovecharán las tecnologías que continúan situando a Apple en una posición predominante en la industria informática, entre las que destacan QuickTime, la arquitectura estándar de ls industria para crear, manipular, jugar y almacenar ficheros multimedia.

Breves

MICROSOFT SELECCIONA A PC DOCS COMO MEJOR SOLUCIÓN DE GESTIÓN DOCUMENTAL

Este es el primer año en el que la compañía Microsoft muestra cómo sus proveedores de soluciones responden a las necesidades de la industria de gestión documental. Entre los finalistas se incluían FileNet Corporation, Kainos Software Limited y PC DOCS.

Por segunda vez, en 1997, Microsoft elige a PC DOCS como la mejor solución de gestión documental basada en su tecnología. Anteriormente, PC DOCS fue el único fabricante de gestión documental que quedó finalista en la categoría de mejor desarrollador de soluciones para Microsoft.

ACUERDO ENTRE LAS EMPRESAS EDS E INTERGRAPH PARA LIDERAR EL MERCADO DE CAD/CAM

Las compañías EDS e Intergraph han formalizado un acuerdo de intenciones para construir una compañía que dé respuesta a la creciente demanda de soluciones de diseño industrial y fabricación, sobre plataformas Windows, que existe actualmente en el mercado informático mundial.

No se ha llegado aún a la a la aprobación final de este proyecto, puesto que está pendiente de su aceptación por parte de los consejos de dirección de ambas compañías.

EDS tendrá la mayoría de acciones de la nueva compañía, y John Mazzola, actualmente presidente de EDS-Unigraphics será su presidente.

La empresa dedicada al software en C++ y JAVA, ILOG, firma un acuerdo de colaboración con la española Tecsidel

ILOG es líder mundial en la comercialización de componentes software en C++ y JAVA para aplicaciones gráficas y de optimización de recursos. Los productos de ILOG proporcionan al desarrollador bibliotecas de objetos para construir sistemas basados en restricciones para la optimización de recursos, scheduling, logística, planificación y dimensionamiento de recursos; herramientas para construir interfaces gráficas intuitivas en 2D y 3D; sistemas de reglas dinámicas para crear agentes inteligentes y sistema de control de flujos de datos en tiempo real, y servicios para integrar módulos con fuentes de datos relacionales y en tiempo real.

La actividad de la empresa española Tecsidel se centra principalmente en el desarrollo e integración de soluciones informática a medida. La empresa se organiza en torno a cuatro áreas de actividad: Peaje y Control de vehículos, Telecomunicaciones, Sistemas de Información y Logística, siendo sus sectores habituales de trabajo las telecomunicaciones, los servicios, el transporte, la distribución, la industria en general, la banca, las finanzas y la Administración.

Digital integrará en su plataforma Prioris la solución SBS de Microsoft diseñada para PYMES

Digital integrará en sus servidores Prioris MMX 6200, basados en la plataforma x86, la solución BackOffice Small Business de Microsoft, especialmente diseñada para las pequeñas y medianas empresas. Esta nueva solución ofrece a las PYMES, a un precio asequible, las soluciones más avanzadas en comunicaciones, así como acceso a Internet. Asimismo, su fácil manejo y mantenimiento permite a los usuarios centrarse en sus oportunidades de negocio, sin preocuparse del control de la infraestructura tecnológica.

Por su parte, la plataforma Prioris 6200 de Digital está preparada para satisfacer las necesidades departamentales de cada empresa.

DELPHI, C++ BUILDER Y VISUAL DBASE

Los cursos del Soporte Técnico Borland

Borland ha desarrollado una serie de cursos que están basados en el conocimiento diario de las dudas y dificultades que se presentan a los usuarios en cada uno de los productos de esta casa.

La empresa ha diferenciado en tres bloques estos cursos. Los primeros son los "presenciales", que se imparten en las instalaciones de la empresa en Madrid y Barcelona. El segundo bloque es el de "empresa", impartidas en las instalaciones de sus clientes. Por último, está la modalidad de "a distancia" que consta de entregas de unidades didácticas, ejemplos y ejercicios. Una vez finalizado el temario, recibirá una prueba de evaluación, que tendrá que cumplimentar.

En todos los productos de Borland se incluye una suscripción anual al Soporte de Desarrollo para, de esta manera, hacer el seguimiento de las dificultades en la aplicación práctica de los conocimientos obtenidos.

Integración de la arquitectura Microsoft Windows DNA Y COM+ con el entorno corporativo SAP R/3

Microsoft Corporation y SAP AG, suministrador líder de aplicaciones corporativas, han anunciado hoy su colaboración para la integración entre la nueva arquitectura Windows DNA (Distributed interNet Applications) de Microsoft y COM+, una extensión de COM, con el Business Framework de SAP. Esta iniciativa es la última de una estrecha relación entre SAP y Microsoft para desarrollar conjuntamente interfaces comunes, que conduzcan las transacciones corporativas a través de Internet.

Asimismo, Microsoft y SAP están realizando acercamientos con sus respectivas arquitecturas para que éstas sean compatibles. M. Windows DNA junto com COM+ es un nuevo entorno de trabajo para soluciones informáticas distribuidas multinivel, escalables y modernas que trabajen sobre cualquier red. Esta arquitectura complementa al Business Framework de SAP, que ofrece una arquitectura abierta basada en estándares para integrar tanto los módulos corporativos de R/3 como los módulos que no ofrece SAP.

Unisys y Microsoft unidos por Windows NT

Unisys y Microsoft Corporation han anunciado una iniciativa conjunta para ofrecer a sus clientes soluciones de misión crítica basadas en el sistema operativo Microsoft Windows NT Server y en la plataforma corporativa de Microsoft. Según el acuerdo, Unisys formará y certificará a más de 2.000 profesionales en los productos corporativos de Microsoft y establecerá cinco centros de desarrollo de aplicaciones dedicados a las soluciones de Microsoft.

Unisys y Microsoft se alían para optimizar el funcionamiento del software en los servidores de altas prestaciones de Unisys y para ofrecer soluciones corporativas para la administración y para empresas de servicios financieros. Además, Unisys utilizará a nivel interno los productos Microsoft Exchange Server y Microsoft Internet Information Server.

Las dos empresas combinarán la plataforma corporativa de Microsoft con los servicios de información, las tecnologías y el soporte global de la compañía Unisys para dar respuesta a la creciente demanda de soluciones de tipo corporativo: soluciones de alta escalabilidad, disponibilidad, fiabilidad y manejabilidad, que ejecuten los procesos de negocio básico de las empresas.

Claris Home Page 3.0 para Windows 95, un nuevo competidor en el campo del diseño de páginas Web

Claris Corp., ha anunciado la disponibilidad de la nueva versión de su programa editor de páginas web, Claris Home Page 3.0 para Windows 95, Windows 3.1, Windows NT y Mac OS 8. Una sencilla herramienta de autor para la Web, que permite, a los desarrolladores de cualquier nivel, diseñar, publicar, y gestionar sus propias páginas web de forma sencilla y rápida.

El programa incluye novedades tales como: los Asistentes, ocho en total, que guiarán al usuario a través de todo el proceso de creación haciéndose cargo, automáticamente, de la gestión del complejo trabajo de programación HTML. Añade 45 plantillas y 18 estilos de diseño profesional, para dar a su especio web un toque más personal. La conectividad con bases de datos FileMaker Pro 4.0, ahora será más fácil añadir formularios interactivos para captar datos de los visitantes, acceder a búsquedas en bases de datos y demás acciones. Las nuevas herramientas de gestión de espacios (Site Editor) ofrecen un esquema completo de las páginas y los contenidos de su espacio, así como permiten identificar y repara los enlaces rotos. También verifican, automáticamente, los *anchors* de las páginas.

Claris Home Page 3.0 está dirigido a las Pymes que quieran competir con grandes cuentas pero que no dispongan de un desarrollador profesional en la plantilla. A los usuarios domésticos que desarrollan espacios web particulares por interés o como hobby. También a los profesores y educadores que enseñan cómo desarrollar páginas web, que promuevan las actividades del centro o que faciliten información a los padres vía Web.

Claris Home Page 3.0 está disponible en el mercado español a partir del mes de noviembre. Los requisitos del sistema para la plataforma Windows son: procesador 486 o superior, lector de CD-ROM 8 Mb. de RAM (16 Mb. para NT). Y para la plataforma Mac: 68020 o superior, sistema 7.1 o superior, lector de CD-ROM, 8 Mb. de RAM.

Breves

DIGITAL E INTEL LLEGAN A UN ACUERDO

Digital Equipment Corporation e Intel Corporation han acordado establecer una relación comercial amplia que proporcionará significativos beneficios a corto y largo plazo a ambas compañías y a sus clientes.

El acuerdo que abarca varios años incluye la venta de las operaciones de producción semiconductor de Digital a Intel por aproximadamente 700 millones de dólares, licencias cruzadas de patentes, suministro de microprocesadores Intel y Alpha y desarrollo de futuros sistemas basados en los microprocesadores 64 bits de Intel.

SCOTT MCNEALY, PRESIDENTE DE SUN MICROSYSTEMS EN MADRID

El presidente de Sun Microsystems ha visitado por primera vez España para exponer su polémica visión sobre el futuro de las Tecnologías de la Información y la industria que se está creando alrededor de Java. Durante su ponencia ante el club de Usuarios de Tecnologías de la Información el pasado 4 de noviembre, Scott McNealy analizó el presente y el futuro del sector y las implicaciones económicas de su evolución.

Sun Microsystems cuenta con una facturación anual superior a 8.598 billones de dólares, su singular visión "The Network is the Computer" ha posicionado a esta compañía como uno de los principales proveedores mundiales de soluciones, orientadas a soporte de aplicaciones de misión crítica, Intranet e Internet, donde Sun Microsystems lidera el sector.

Edición Especial Microsoft Word 97

El autor Ron Person, de la mano de QUE lanza al mercado un exhaustivo manual sobre el procesador de textos de Microsoft Word 97.

Este libro muestra toda la potencia de la última versión del editor de textos Word. Combina de forma sencilla las funcionalidades más básicas con las más avanzadas permitiendo en todo momento al lector ir aprendiendo de forma gradual.

El manual está estructurado en nueve capítulos diferentes. Cada uno de ellos se centra en un tema específico del proceso de textos. A medida que el usuario va avanzando en los temas, avanza, también, la dificultad de los mismos.

El autor se ha hecho eco de la repercusión que está teniendo el mundo de la red de Internet en la informática actual y, por ello, en los capítulos se muestra cómo se pueden realizar páginas Web de forma rápida y eficaz, sin tener conocimientos previos del lenguaje HTML.

Por otra parte, a lo largo del manual se descubre la forma de convertir documentos sencillos en documentos elaborados, mediante la utilización de gráficos e imágenes. Al mismo tiempo, permite aprender la forma de modificar gráficos utilizando sus últimas herramientas.

Hay que destacar el elevado número de imágenes con las que cuenta el libro (capturas de pantallas de Word 97) que ayudan al lector a situarse dentro de la dinámica del procesador facilitando mucho la labor del usuario. Siguiendo la misma tónica, en las páginas finales del libro se ofrece un índice temático que atiende y responde, sobre todo, a su calidad de manual, y donde se puede acceder de una forma rápida a temas determinados de interés en un momento dado.

Su autor, Ron Person, ha escrito más de 2º libros, incluyendo el *bestseller* Edición Especial Microsoft Excel 97. Ha sido autor líder de dos bestsellers sobre Windows 95, Special Edition Using Windows 95 y Platinum Edition Using Windows 95.

Editorial: Prentice Hall, 1997
Nº de páginas: 1.000
PVP: 8.990 Ptas.

Autor: Ron Person y Karen Rose
Idioma: Castellano
Nivel: Principiante-Avanzado

El Entorno de Aplicaciones Forté

HERRAMIENTAS DE DESARROLLO

Bonifacio Villalobos

Estas nuevas aplicaciones críticas se están desarrollando e implantando en entornos distribuidos complejos que contienen navegadores Web, así como interfaces gráficas de usuario, servidores de aplicación paquetizados o a medida, una gran variedad de servicios de datos y productos de comunicaciones heterogéneos. Aislar a los desarrolladores de estas mil interfaces de programación es un requisito crucial para la productividad del desarrollador y la facilidad de mantenimiento del sistema.

La implantación y gestión de la explotación de estas aplicaciones de carácter corporativo supone un reto para la gran mayoría de departamentos de TI. Estos han realizado con éxito implantaciones de aplicaciones críticas en entornos centralizados y requieren de los entornos distribuidos un nivel de soporte similar a la hora de instalar y explotar los sistemas y aplicaciones distribuidos. La mayor complejidad de estos sistemas hace indispensable el contar con las herramientas adecuadas para asegurar su correcto funcionamiento día a día.

Se necesita una nueva categoría de herramientas para el desarrollo, implantación y explotación de esta nueva generación de aplicaciones. El Entorno de Aplicaciones Forté, que es el objeto de este artículo, cubre esta necesidad.

Para el desarrollador, Forté proporciona un conjunto completo de herramientas orientadas a objeto que soportan el desarrollo de los componentes, tanto cliente como servidor, de las aplicaciones.

Forté incluye un generador de aplicaciones, un diseñador de interfaces gráficas de usuario, un generador de páginas HTML y applets Java, un lenguaje orientado a objeto de cuarta generación, un conjunto completo de librerías de clases y un repositorio para el soporte a equipos de desarrollo. Forté está completamente integrado con los principales sistemas de gestión de bases de datos relacionales (SGBDRs) y puede interaccionar con sistemas externos como aplicaciones existentes y dispositivos electrónicos. Forté también incorpora un diseñador de procesos de negocio dirigido a la construcción de aplicaciones que estén basadas en flujos de trabajo.

Forté proporciona un conjunto completo de herramientas orientadas a objeto que soportan el desarrollo de los componentes, tanto cliente como servidor, de las aplicaciones.

Un entorno empresarial muy dinámico, conjuntamente con unas tecnologías de la información en rápida evolución, está dando lugar a una nueva generación de aplicaciones corporativas.

Para la implantación, Forté genera un sistema completo y optimizado para el entorno de instalación determinado. Incluye un entorno de ejecución distribuida robusto, con mecanismos sofisticados de rendimiento, escalabilidad y fiabilidad. También incorpora un conjunto de herramientas de gestión de aplicaciones para explotar las aplicaciones instaladas. Automatiza la infraestructura que las aplicaciones requieren, permitiendo a los desarrolladores responder con mayor agilidad a los requisitos del negocio.

Figura 1. El entorno Forté.

Componentes principales

Forté consta de los siguientes componentes individuales:

- **TOOL.** Es un lenguaje de programación de cuarta generación, totalmente orientado a objeto. Viene acompañado de un depurador simbólico multitarea a nivel de código fuente.

- **Librerías.** Son conjuntos de clases prefabricadas que proporcionan la infraestructura básica para la construcción de las aplicaciones. Las clases de Forté definen componentes básicos de la aplicación, como ventanas, menús o campos; así como servicios externos, gestión de accesos a bases de datos y gestión de transacciones.

- **Las Herramientas o workshops.** Conjunto de herramientas gráficas para el desarrollo de la aplicación, la construcción de la interfaz gráfica de usuario, el particionamiento y la distribución de la aplicación a los servidores físicos.

- **Repositorio.** Servidor centralizado de almacenamiento que soporta totalmente el trabajo en grupo de los desarrolladores, incluidos la compartición de código y el versionado.

- **Express.** Es la herramienta de generación de aplicaciones. Utilizando las herramientas de Express, el editor de Modelo de Negocio y el editor de Modelo de Aplicación, es posible construir modelos gráficos de las clases de objetos de la aplicación y del flujo de ventanas de la misma. Forté Express genera de forma automática una aplicación en base a los modelos definidos. Además, es posible utilizar el resto de herramientas de Forté para refinar y completar la aplicación, por ejemplo con reglas de negocio específicas.

- **Conductor.** Es la herramienta de Forté para el desarrollo, instalación y explotación de aplicaciones basadas en flujos de trabajo. Conductor proporciona un conjunto de herramientas para construir la lógica de los procesos, definiciones de proceso de Conductor ejecutadas por un motor de procesos de Conductor. También proporciona un conjunto de APIs mediante las cuales las aplicaciones cliente, que ejecutan lógica de aplicación (actividades) en un proceso, pueden comunicarse con el motor. La gestión de los motores y la monitorización de los procesos de negocio también se realizan mediante herramientas proporcionadas por Conductor.

- **Consola.** La consola permite la definición de los entornos de ejecución de las aplicaciones Forté, así como la monitorización y el ajuste (*tuning*) del rendimiento de las aplicaciones en explotación.

El Lenguaje de Forté: TOOL

TOOL es el lenguaje de programación en el entorno de aplicaciones Forté. TOOL es un lenguaje de propósito general, orientado a objeto. Es decir, posee los mecanismos de encapsulación, herencia (simple) y polimorfismo. El acrónimo TOOL significa *Transactional Object-Oriented Language* y pretende reflejar el hecho de que TOOL, además de los mecanismos propios de un lenguaje 4GL-OO de última generación como gestión automática de memoria,..., contiene toda una serie de primitivas que permiten al usuario explotar las capacidades de la Arquitectura de Ejecución del entorno.

Los conceptos más importantes enmarcados dentro de la filosofía de TOOL son los siguientes:

- **Clases y Objetos.** Una clase es la definición de la estructura y comportamiento de un objeto. Un objeto es la representación lógica de algo significativo para la aplicación; por ejemplo un pedido, un cliente, o una reclamación. La clase define los datos y las operaciones de todos los objetos correspondientes a dicha clase. Las clases en Forté contienen atributos, que almacenan los datos relativos al objeto; métodos, que se usan para manipular el objeto y le dotan de funcionalidad, y eventos, que son notificaciones que provocan acciones por parte del objeto.

- **Encapsulación.** Puesto que en Forté lo único que puede operar sobre un objeto son sus métodos, sólo éstos necesitan conocer la estructura interna del objeto. Limitando el acceso y el conocimiento de dicha estructura interna, es como la encapsulación aísla a la aplicación de cambios que pudieran ocurrir en el objeto. Cuando se producen dichos cambios, sólo es necesario modificar los métodos; manteniendo, eso sí, la definición de la interfaz de los mismos.

- **Herencia.** La herramienta soporta totalmente el mecanismo de herencia. De este modo, las subclases heredan los atributos, métodos y eventos (y sus correspondientes manejadores de eventos, que realizan las acciones en respuesta a los eventos). Cuando se define una subclase, los métodos heredados de su superclase pueden ser sustituidos por métodos específicos de la subclase.

Forté también permite la "sobrecarga" de métodos; es decir, una clase de objetos pueden tener varios métodos con el mismo nombre, pero con diferentes codificaciones que se distinguen por los argumentos que acepta. Automáticamente invoca al método correcto en base al tipo de datos que se le pasa como parámetro.

- **Polimorfismo.** Como el lector sabrá, el polimorfismo indica la capacidad de invocar diferentes implementaciones del mismo método dependiendo de la clase a la que pertenece el objeto llamado. Forté da un soporte completo al polimorfismo además de permitir la sobrecarga de métodos, como ya indiqué.

Las sentencias de programación del lenguaje TOOL pueden encuadrarse en las siguientes categorías generales:

- **Declaración y asignación.** Las sentencias de declaración y asignación permiten declarar variables locales y asignarles valores.

- **Control de flujo.** Son las sentencias usuales de control del flujo de ejecución e incluye las sentencias *case, for*, *if* y *while*.

- **Gestión de eventos.** La sentencia *post* dispara un evento. La sentencia *event* procesa la respuesta a uno o más eventos específicos. La sentencia *register* incluye manejadores de evento con nombre dentro de una sentencia *event*.

- **Multitarea.** La sentencia *start task* inicia una nueva tarea. La sentencia *shutdown* finaliza la tarea desde donde se ejecuta.

- **Transacciones.** El bloque *begin transaction/end transaction* ejecuta el código que engloba como una única transacción, manteniendo su atomicidad.

- **Acceso a base de datos.** Las sentencias de SQL estándar manipulan tablas de bases de datos relacionales. La sentencia *sql execute inmediate* permite ejecutar cualquier sentencia SQL, incluidas las extensiones propias de cada gestor, y *sql execute procedure* permite ejecutar un procedimiento almacenado en cualquier gestor.

- **Manejo de excepciones.** La cláusula *exception* permite la captura de excepciones en el bloque de sentencias actual. La sentencia *raise* genera una excepción.

Las Librerías de Forté

Forté proporciona un conjunto completo de librerías que permiten acceder a los recursos de cada plataforma bajo una interfaz única que aísla las aplicaciones de las diferencias existentes entre las plataformas soportadas.

- **FrameWork.** Es la librería básica de Forté. Proporciona soporte a la infraestructura de Forté: tareas, eventos, transacciones, excepciones, applets,etc.

- **Display.** Contiene los componentes para la construcción de interfaces gráficas de usuario: ventanas, menús, campos, botones y otros widgets.

- **GenericDBMS.** Esta librería incluye las clases que realizan la conexión a bases de datos, gestión de sesiones, bloqueos, SQL, caché, y acceso a librerías de bases de datos específicas.

- **OLE.** Como su nombre indica esta librería permite la conexión en ambos sentidos entre aplicaciones Forté y otras aplicaciones que también dispongan de interfaz OLE.

- **DCE.** La integración de aplicaciones Forté con otro tipo de sistemas abiertos se realiza mediante esta librería, que implementa protocolos RPC y DCE.

- **ForeignObjMgr.** Forté puede interaccionar con sistemas de objetos distribuidos basados en estándares CORBA 2 e IIOP. El soporte se encuentra en las clases de esta librería.

- **SystemMonitor.** Forté incluye de forma estándar un conjunto de agentes de monitorización de aplicaciones. Si queremos un control más "a medida" podemos utilizar esta librería que contiene las clases necesarias para desarrollar nuevos agentes.

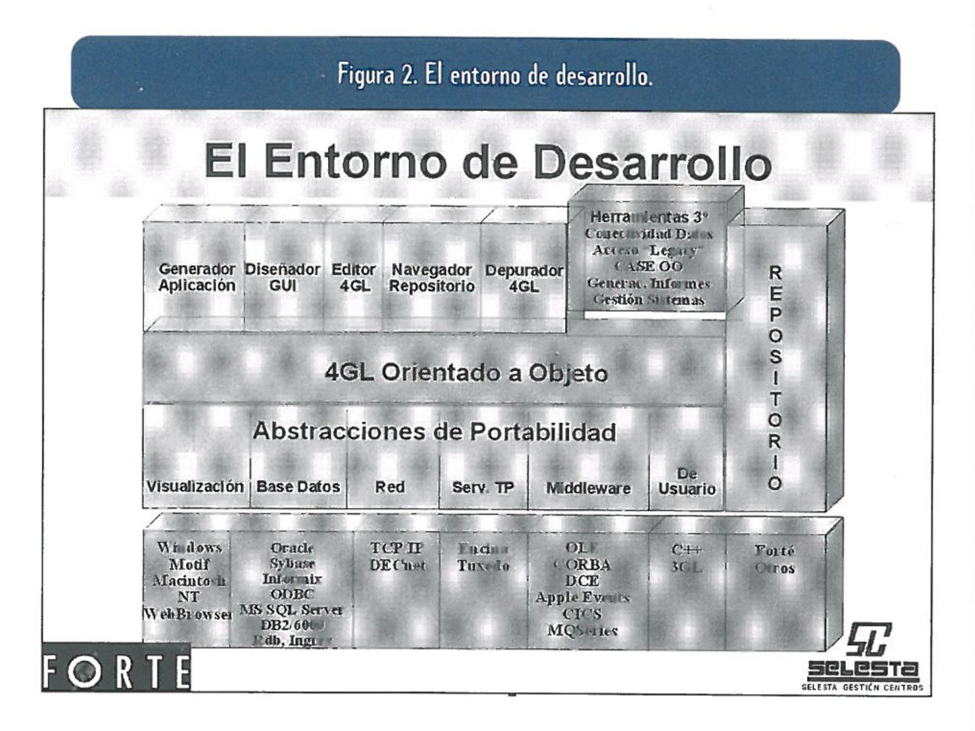


Figura 2. El entorno de desarrollo.

Herramientas del Entorno de Aplicaciones Forté

En el Entorno de Aplicaciones tenemos tres grupos de herramientas que se corresponden con las tres fases principales en el ciclo de vida de un sistema: Desarrollo, Implantación y Explotación.

- **Herramientas de Desarrollo**

El conjunto de herramientas de desarrollo de Forté forman en sí mismas un entorno cliente/servidor. El servidor está constituido por el Repositorio, mientras el resto de herramientas residen en las estaciones de trabajo. La base del entorno de desarrollo es el lenguaje TOOL que ya hemos descrito. El lenguaje se apoya en el conjunto de librerías mencionado a fin de soportar múltiples plataformas con idéntica funcionalidad.

- **Repository Workshop.** Permite a cada programador crear su "espacio de trabajo", de este modo cada programador trabaja de forma independiente al resto de programadores pero sin mermar las facilidades de colaboración. Cualquier cambio que un programador realice en su espacio de trabajo será visible para el resto de programadores cuando lo integre en el proyecto. Cualquier cambio realizado por otros programadores no es visible para mí hasta que no sincronizo los cambios en mi espacio de trabajo con la *baseline* del proyecto.

  Comandos específicos de control de código permiten realizar *check-out* del cualquier componente del proyecto, dando a un programador acceso exclusivo en escritura hasta que se integre de nuevo en el repositorio.

  La *Repository Workshop* es también el centro de control de las herramientas de Forté. Desde ella es posible acceder a cualquier herramiena y examinar o modificar componentes o configuraciones del código del proyecto.

- **Project Workshop.** En la terminología de Forté, un "proyecto" es una definición de aplicación, una definición de servicios, o una librería de componentes reutilizables. Cuando se trata de una definición de aplicación, define la interfaz gráfica de usuario y especifica la lógica de la aplicación. Cuando el proyecto es una definición de servicios especifica la lógica de programación del servicio. Lógica que puede ser compartida por un conjunto de aplicaciones. Cuando el proyecto es una librería, sólo proporciona las definiciones de sus componentes, para que puedan ser utilizados en otros proyectos; ocultando la codificación de los mismos.

- **Class Workshop.** Esta herramienta permite la creación de clases específicas de aplicación dentro de un proyecto determinado. Para ello es necesario construir todos los elementos que conforman una clase en Forté: atributos, métodos, eventos, manejadores de eventos y constantes.

- **Interface Workshop.** Una interfaz define un conjunto de elementos de una clase sin especificar el código que los implementa. Estos elementos son: atributos virtuales, métodos, eventos, manejadores de eventos, constantes de la interfaz. La interfaz proporciona la cabecera del método o el manejador de eventos que definen una interfaz estándar de un objeto. El código de dicho método o manejador lo proporciona la clase que implementa la interfaz.

- **Windows Workshop.** Esta herramienta es un editor de ventanas similar al utilizado en otros entornos para el desarrollo de interfaces gráficas de usuario. Consta de un área

de trabajo en la que se dibuja la ventana y una paleta de controles que contiene los *widgets*, por ejemplo campos, títulos, botones, listas, botones de radio, imágenes, tablas, grids, etc. Los menús de barra se editan con una herramienta particular, *menu workshop*.Una vez dibujada la ventana, la herramienta permite ejecutarla a fin de conocer con precisión su apariencia real.

- **Menu Workshop.** Es un editor gráfico para definir la barra de menús de una ventana y los menús asociados a cada una de las opciones de la barra. Cada menú puede incorporar:

  - Botones, son los controles usuales que visualizan una acción.
  - Marcas, visualizan una marca que el usuario final puede activar o no.
  - Listas, lista de opciones asociada a una entrada de menú.
  - Menús desplazables.

- **Method Workshop.** Es el editor de código TOOL con el que se construyen los métodos asociados a cada clase. Se define el nombre, los parámetros, valores de retorno y se escribe el código TOOL que implementa la lógica del método. La herramienta verifica que el método está correctamente escrito, antes de enviarlo al depurador para la monitorización de su ejecución.

- **Event Handler Workshop.** Muy similar a la herramienta anterior, la *event handler workshop* permite editar los bloques de código TOOL que han de ejecutarse como respuesta a uno o más eventos. Por supuesto también se define la interfaz del manejador, es decir, su nombre y parámetros.

- **Cursor Workshop.** Mediante la cursor workshop podemos definir editar cursores de bases de datos. La definición del cursor consisten en un nombre y una sentencia *select* que selecciona el conjunto de filas de la base de datos sobre las que opera el cursor. Para definirlo especificamos su nombre, el área de datos y escribimos el código TOOL que ejecuta el cursor.

- **Debugger (Depurador).** El depurador de Forté permite trabajar a nivel de código fuente TOOL, sin necesidad de compilar las aplicaciones a depurar. Es posible depurar aplicaciones completas o conjuntos de clases específicos que conformarán una parte de la aplicación. Entre sus características se destaca el hecho de permitir puntos de ruptura asociados a eventos y excepciones, y su capacidad para depurar múltiples tareas en paralelo.

Herramientas de Instalación

Forté ofrece como herramientas de instalación el conjunto formado por una herramienta gráfica, la *partition workshop*, un lenguaje de comandos denominado, *iscript*, y un agente de instalación. Estas herramientas se encuentran integradas con el repositorio de Forté, del cual extraen el código que se instalará en cada plataforma donde se deba ejecutar la aplicación distribuida.

- **Partition Workshop.** Permite examinar y modificar configuraciones, generar y compilar el código a instalar en el entorno. Una configuración es una aplicación con una descomposición en módulos ("particionada" según la terminología de Forté) específica para un entorno concreto. La configuración determina en qué nodo del entorno se ejecuta cada uno de los módulos/"particiones". Es posible por tanto definir e instalar distintas configuraciones para una misma aplicación en función de los diferentes entornos en los que vaya a estar operativa. Este proceso de particionamiento no afecta al código fuente de la aplicación, es Forté el encargado de proveer la infraestructura necesaria para que la aplicación funcione correctamente en cada una de sus configuraciones.

Una vez definida la configuración podemos generar una distribución de la aplicación, en este proceso la herramienta extrae el código TOOL del repositorio y compila cada partición para la plataforma donde deba ejecutarse. El proceso de compilación tiene dos pasos. En el primer paso Forté, a partir del código fuente TOOL, genera código C++ específico de la plataforma. En el segundo paso Forté llama al compilador nativo C++ de la plataforma y compila el código generado; enlazándolo con las librerías propias de Forté.

Una vez lista la distribución, Forté distribuye físicamente los ficheros ejecutables que conforman cada partición a cada una de las máquinas implicadas y los instala. Para ello colaboran la *partition workshop* y los agentes de instalación.

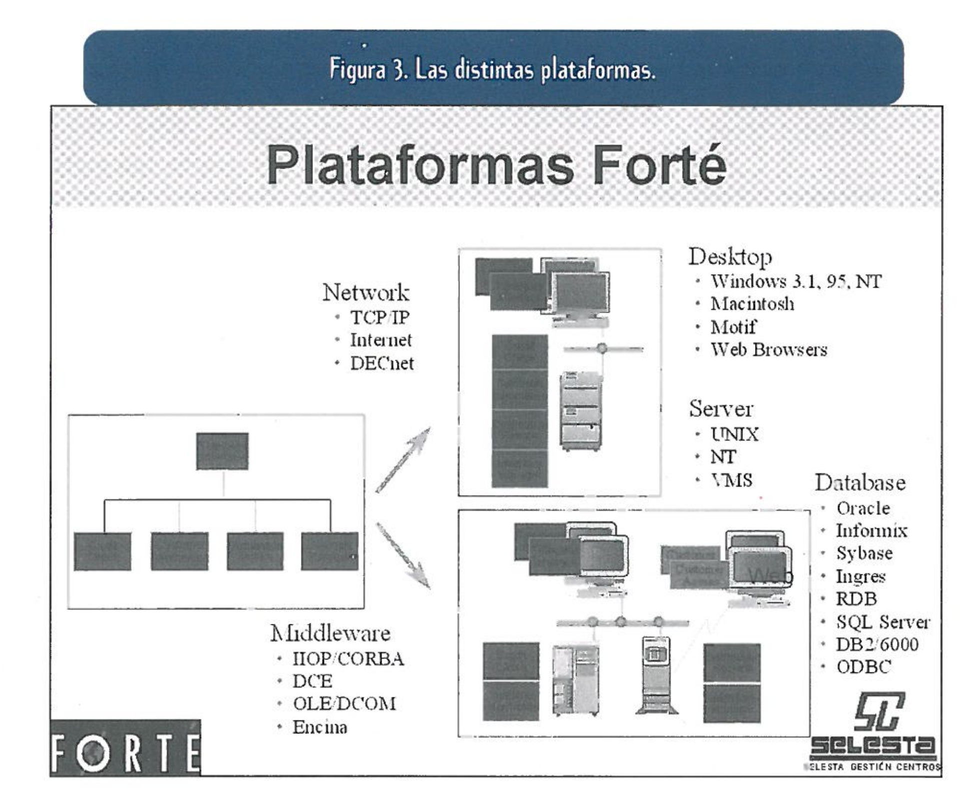
- **Agente de instalación.** Este agente forma parte de la arquitectura de ejecución de Forté. Trabaja a las órdenes de la herramienta de particionamiento o del lenguaje de comandos. Se instala en todas las máquinas que ejecutan aplicaciones Forté, tanto clientes como servidores.

- **Lenguaje iscript de comandos.** Todas las operaciones que realizamos con la herramienta visual de particionamiento descrita anteriormente es posible realizarlas mediante el lenguaje de comandos. Resulta útil para tareas como la generación de distribuciones y su envío a cada una de las máquinas implicadas en horas de baja utilización de las comunicaciones (por las noches, fines de semana) y para automatizar procesos de mantenimiento de las configuraciones de las aplicaciones.

Herramientas de Explotación

Si algo caracteriza al entorno de aplicaciones Forté y lo diferencia claramente de otros entornos de desarrollo es su capacidad para gestionar y monitorizar las aplicaciones. La infraestructura de soporte

Figura 3. Las distintas plataformas.



necesaria para esta gestión y monitorización está embebida en las propias librerías de ejecución del producto, siendo por tanto transparente a los programandores al no exigir sentencias de código extra para incluirla en las aplicaciones.

Desde el punto de vista de un programador puede parecernos que este grupo de herramientas no son imprescindibles y que sus usuarios reales son las personas que componen los departamentos de explotación y sistemas en una compañía. Sin embargo dado que disponemos de las herramientas desde el comienzo del proyecto, es posible utilizarlas para monitorizar todo aquello que vamos desarrollando. Es en este punto dónde nos van aportar una eficaz ayuda a la hora de detectar anomalías, decisiones de diseño incorrectas, código poco optimizado, etc. Todo ello durante el desarrollo, sin esperar a la implantación de la aplicación, ni siquiera en los puestos piloto.

Las herramientas de explotación se componen de una consola de entorno, *environment console*, un lenguaje de comandos denominado *e-script*, un conjunto de agentes y una librería para el desarrollo de agentes.

- **Environment Console.** La consola de entorno es una herramienta gráfica que permite examinar, modificar, y crear entornos Forté de forma centralizada. Un entorno Forté está formado por un conjunto de nodos de una red que ejecutan Forté. La consola presenta el entorno mediante un editor que visualiza el entorno como un mapa de red que incluye todos los nodos incluidos en dicho entorno. Para ver el detalle de un nodo en particular se da doble-click con el ratón sobre el icono del nodo. Para entornos muy complejos la consola puede también suministrar toda la información en modo texto.

Para definir un entorno se añaden los nodos utilizando el ratón mediante *drag&drop*, seleccionando nodos individuales y soltándolos en su entorno. Entonces se completa la ventana de detalle de cada nodo con información relativa a la plataforma software, recursos externos, aplicaciones existentes, etc.

Además de gestionar los entornos Forté, la consola permite monitorizar la ejecución de las aplicaciones desde un único punto. Además de las propiedades de cada nodo del entorno, la consola visualiza las particiones que se ejecutan en cada nodo. Podemos monitorizar el rendimiento de cada partición, monitorizar alertas enviadas por las aplicaciones, y arrancar o detener servicios en cada nodo. La información global del entorno se suministra de forma automática, para ver la información detallada seleccionamos cada nodo, la partición a examinar y accedemos al detalle de su ejecución.

- **Lenguaje e-script de comandos.** La monitorización de las aplicaciones no debe exigir la presencia constante de un operador en la consola, mediante sentencias de comandos del lenguaje *e-script* podemos realizar todas las tareas de monitorización y control del entorno. Es especialmente útil para desarrollar ficheros de comandos para el arranque y/o parada de servicios a horas determinadas, o bien en función de la carga de los mismos, o como parte de los trabajos de explotación de los diferentes sistemas utilizados por la aplicación distribuida.

- **Agentes de Monitorización - Librería de Agentes.** Son los encargados de llevar a cabo sobre cada nodo de un entorno Forté todas las acciones indicadas por la consola o por ficheros de comandos. Asímismo también realizan la monitorización de las aplicaciones en marcha, suministrando información a la consola. Estos agentes son capaces de comunicarse utilizando varios protocolos, el propio de Forté para comunicación con la consola del entorno, protocolo SNMP para comunicación con consolas tipo Openview o Netview, protocolo Tivoli.

No sólo disponemos de los agentes ya incluidos en el entorno, sino que Forté proporciona también la librería de objetos que soporta dichos agentes. Esto permite el desarrollo de nuevos agentes, específicos de apli-

¿TE ATREVES?

esde **Sólo Programadores** estamos preparándonos para un torneo muy especial, de ordenador a ordenador, de programador a programador. El torneo de edrez está empezando a tomar forma y éstas son las bases, aunque también podéis informaros en AWS en la calle Serrano Jover, 3, de Madrid, o en el teléfono 1) 5 42 50 87. Para conocer cuál será el alcance de la participación, os pedimos que os inscribáis enviando vuestros datos a la dirección de AWS o a la dirección e-mail: **solop@towercom.es.** El periodo de inscripción ya ha comenzado y terminará el 30 de noviembre, y la fecha límite para la recepción de los programas el 15 de diciembre de 1997. La celebración del torneo será anunciada con suficiente antelación y podréis participar en persona o desde vuestra propia ciudad, que vuestro programa os representará y en todo momento estará presente un notario.

ASES DEL TORNEO

Se presentará un programa que juegue al ajedrez recibiendo la jugada del contrario y devolviendo la propia. El tiempo de la partida será medido por un juez y la partida se jugará físicamente en una mesa con un tablero siguiendo las instrucciones que cada ordenador dé; cada ordenador avisará de que ha elegido un movimiento con un pitido y mostrará su jugada en la pantalla de la forma: casilla a mover - casilla destino. La notación de la casilla vendrá dada por su columna y su fila. Para las columnas se utilizarán las de la A a la H siendo la columna A la primera columna de la izquierda, visto el tablero desde la posición de las blancas. Para las filas utilizaremos los números del 1 al 8 siendo la fila 1 la más cercana a nosotros (visto el tablero desde el lado de las blancas). Como ejemplo, el movimiento del peón de rey dos lugares hacia adelante sería: "E2 - E4". La jugada que hace el contrario se anotará en el ordenador de forma manual tecleando "E2-E4" <RETURN>.

El programa jugará al ajedrez según las normas de este juego, admitiendo todos sus movimientos.

El programa se ejecutará bajo Windows 95, MS Dos 6.22 y Linux, en un Pentium 166 con 16 MB de RAM.

El programa podrá jugar con las blancas o con las negras indistintamente.

Se debe entregar el código fuente, la versión del compilador utilizado y una breve documentacion para su utilización.

Un programa podrá ser descalificado si realiza un movimiento no válido y esto deberá ser advertido por el contrario; por lo tanto, un movimiento será válido siempre que el otro programa no nos advierta de lo contrario. De la misma forma, si un programa nos advierte de un movimiento erróneo y éste no lo es, será descalificado.

Los jueces son los encargados de advertir las posiciones de tablas, el programa no advertirá de ello.

El programa debe detectar si ha ganado la partida y quedará descalificado ante cualquier detección errónea. No se avisará al contrario en el caso de que deje al rey al descubierto y ganará la partida el que antes mate al rey.

La metodología del concurso se reservará hasta que sea conocido el número de participantes (eliminatoria, puntos, etc.)

). No se podrán utilizar en el programa bases de datos adicionales.

l. Los derechos del programa ganador pasarán a la propiedad de la empresa Tower Communications, SRL.

2. El premio consistirá en un Pentium PRO 200 con 64 MB RAM, VIRGE4 MB, CDX 16, disco duro 5 GB, monitor 17", teclado, ratón y disquetera.

NEXO A LAS BASES

La velocidad de juego se establece en 2 horas (1 hora por jugador).

Las jugadas que el árbitro considere erróneas eliminarán al programa que las haya indicado.

El programa debe ser original.

La fecha de recepción del fuente y el compilador se establece entre el 1 y el 31 de Enero. Se deberán remitir a la siguiente dirección: AWS Informática, Serrano Jover 3, 28.015, Madrid, a la atención de Fernando Núñez.

El primer día de celebración del torneo será el 8 de Febrero de 1998 en la dirección indicada en el punto anterior.

Debido a las peticiones recibidas por los participantes, se admiten bases de datos adicionales.

CONCURSO DE PROGRAMACIÓN

SÓLO PROGRAMADORES Y AWS INFORMATICA

SERRANO JOVER 3, MADRID

cación, capaces de proporcionar información sobre la actividad de negocio de cada aplicación y de actuar sobre las mismas.

La Arquitectura de Ejecución de Forté

La arquitectura de ejecución de Forté constituye el núcleo del entorno y le aporta sus principales características. Es una nueva generación de arquitecturas que supera tanto la arquitectura de ejecución centralizada típica de los *mainframe* como la arquitectura cliente/servidor de dos niveles propia de las aplicaciones departamentales en entornos cliente/servidor. No obstante se incorporan conceptos tanto de una como de la otra.

La arquitectura cliente/servidor departamental se caracteriza por su rápida evolución tecnológica

De la arquitectura de ejecución centralizada se toman tres características, que por siempre supuestas no dejan de ser críticas, a saber su nivel de fiabilidad, su escalabilidad para soportar grandes cargas de trabajo y su capacidad para ser gestionada. Estas tres características son la base de una arquitectura que deba soportar aplicaciones críticas y corporativas.

La arquitectura cliente/servidor departamental se caracteriza por su rápida evolución tecnológica, una plataforma con múltiples suministradores, interfaces de usuario amigables y herramientas de desarrollo con mayor facilidad de uso. Por otra parte carece de la fiabilidad, escalabilidad y gestionabilidad de los entornos centralizados.

Y ese es precisamente el objetivo de la arquitectura de ejecución de Forté, aunar en un único entorno de aplicaciones herramientas de desarrollo con la capacidad tecnológica y facilidad de uso de los entornos departamentales conjuntamente con los recursos de arquitectura propios de un entorno crítico, es decir, un tipo de soporte específico para un alto nivel de fiabilidad, una fuerte capacidad de escalabilidad de las aplicaciones y unos mecanismos de gestión de entorno potentes. Todo esto llevó al desarrollo de una arquitectura de ejecución propia, basada en objetos distribuidos, capaz de cumplir estos objetivos.

La Arquitectura de Ejecución para sistemas distribuidos

Anteriormente hemos mencionado características que consideramos básicas para una arquitectura de ejecución distribuida. En este apartado insistiremos en los aspectos específicos de desarrollo que debe tener el entorno soportado por dicha arquitectura.

Uno de los puntos clave en el que han impactado tecnologías como cliente/servidor o Internet ha sido el replanteamiento de las formas de acceso de los usuarios a las aplicaciones. Si algo ha quedado claro, ha sido precisamente que dichas formas de acceso han tenido una evolución importante y la seguirán teniendo. Ello nos lleva a un primer requisito inexcusable, el entorno de aplicación debe permitirnos tanto en desarrollo como en ejecución diferenciar claramente la interfaz de usuario del resto de la aplicación. Con ello evitaremos que la rápida evolución de dichos interfaces impacten de forma brutal, como lo han hecho hasta ahora, en el mantenimiento de las distintas aplicaciones.

La productividad en desarrollo nunca será suficiente, a mayor demanda cubierta, mayores expectativas generadas; eso lo sabemos todos muy bien. Resulta muy importante, en consecuencia, el disponer de un entorno de desarrollo que facilite una productividad alta y un buen método para conseguirla es centrar a los programadores en el desarrollo de las funciones de negocio, evitándoles la programación de bajo nivel aumentando así el rendimiento.

Estrechamente relacionado con el requisito anterior está la necesidad de poder distribuir libremente entre los diferentes nodos de un sistema distribuido las funciones de negocio. Implícitamente se está pidiendo una independencia del código respecto a la plataforma tecnológica que lo ejecuta y una independencia, también, respecto a la configuración física final dada a dicha aplicación, es decir, respecto a la distribución de funciones por los nodos.

Consecuentemente el funcionamiento de las aplicaciones no debe depender de dónde se están ejecutando, lo que significa que la localización de cada una de las funciones debe ser transparente de cara al resto de la aplicación.

El disponer de un único lenguaje, soportado en todas las plataformas, facilita todos los puntos señalados anteriormente, siendo uno de los elementos clave a considerar al evaluar un entorno distribuido de aplicaciones.

Por último destacar dos factores que inciden especialmente en los sistemas distribuidos, el primero es la característica dinámica de los mismos, que implica la actualización de los mismos sin interferir con su funcionamiento. El segundo es su crecimiento, convertido en factor crítico en entornos como Internet y que debería basarse para dicha escalabilidad en plataformas estándares de mercado.

Capacidades principales de la Arquitectura

- *Multinivel.* El disponer de una arquitectura de aplicación en varios niveles es un requisito imprescindible para conseguir un sistema escalable. Una arquitectura multinivel real implica disponer de servidores de aplicación puros, es decir, sin gestor de datos, permitiendo además la interconexión entre ellos. De forma que los servidores puedan ser a su vez clientes unos de otros.

  Forté introduce un elemento diferencial en este requisito y es su capacidad para dotar a una misma aplicación de diferentes configuraciones físicas, es decir diferentes distribuciones en niveles sin que por ello deba modificarse el código de la aplicación. Esta facilidad se hace patente en organizaciones grandes, muy distribuidas, en las que por ejemplo se diferencian diferentes entornos de oficinas según su tamaño.

- *Mensajería asíncrona* .¿Cómo se comunican los distintos nodos? La tendencia se dirige hacia sistemas basados en mensajes, por la flexibilidad de configuración que ofrecen y la baja cohesión entre módulos que exige. La mensajería asíncrona, como la proporcionada por Forté, permite evitar el bloqueo de los clientes por saturación de trabajo de los servidores, la realización simultánea de peticiones a diferentes servidores esperando sólo por el más lento y el trabajo en modo síncrono cuando la aplicación lo requiere.

- *Eventos de Negocio.* La programación por eventos, causada por las interfaces gráficas de usuario, irrumpió con fuerza y numerosos problemas a principios de los 90. Hasta ahora se reducía al ámbito de las GUI. Sin embargo, con Forté es posible definir los eventos a nivel de aplicación, sin ligazón alguna a interfaz de usuario. Estos eventos de aplicación tienen una naturaleza distribuida, se producen en una máquina y se reciben en otras siguiendo el mecanismo denominado "publicar y suscribir".

Forté proporciona capacidad para dotar a una misma aplicación de diferentes configuraciones físicas

  Mediante este mecanismo, un módulo publica un evento de aplicación que ha tenido lugar (*post*) y Forté se encarga de hacerlo llegar junto con su información a todos los módulos que se han suscrito al evento (*when*) allá donde se encuentren. Todo sucede en tiempo real y no se usan técnicas de *broadcasting*. Mediante eventos es posible sincronizar el trabajo de los diferentes módulos, y de los usuarios que los utilizan. Los eventos forman la base de "*conductor*", la librería de flujo de trabajo de Forté.

- *Transacciones.* El concepto de transacción todos los conocemos y lo usamos a diario en nuestras aplicaciones. Las transacciones han estado hasta ahora cincunscritas a los gestores de bases de datos y a los monitores de transacciones. Respecto a los gestores de bases de datos, sólo una pequeña fracción de las aplicaciones, los procedimientos almacenados y disparadores, funcionan bajo el control del gestión y el ámbito de la transacción se extiende a ellos. El monitor de transacciones nos da una cobertura más amplia, a nivel de servicio de aplicación (incluso a nivel conversacional en entornos centralizados), a costa de una mayor complejidad de desarrollo y sobre todo de implantación y explotación de las aplicaciones.

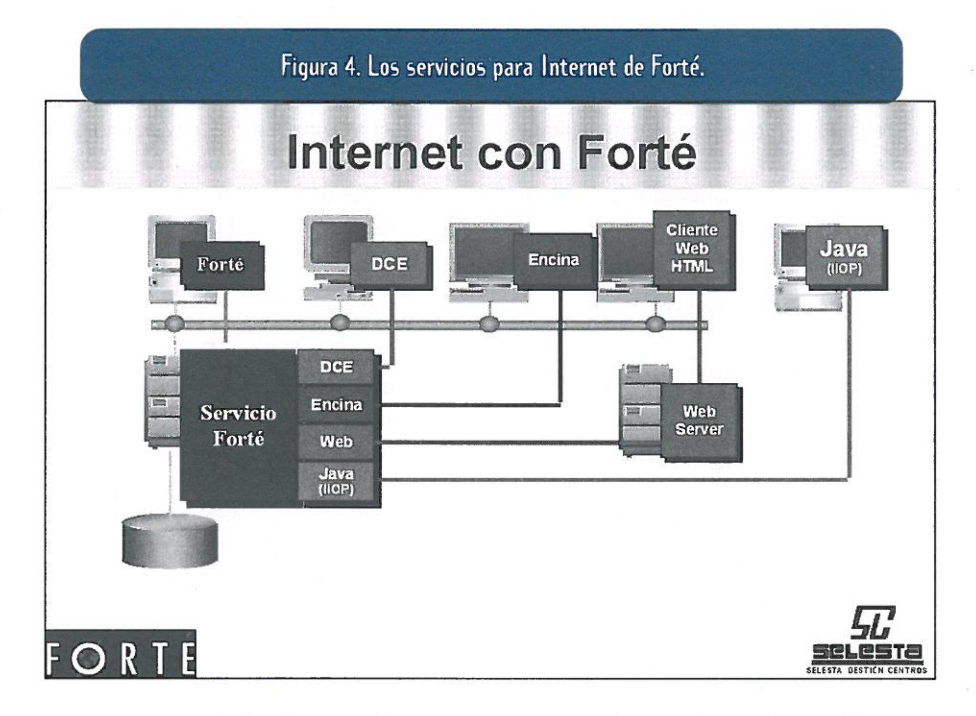
  Una transacción Forté puede incluir cualquier tipo de objeto (incluidos objetos de la interfaz de usuario) en cualquier nodo. Forté automáticamente guarda la información de estado de cualquier objeto transaccional involucrado para restaurarlo si la transacción fallase. Forté también se encarga de enviar el *commit* o el *rollback* al gestor de base de datos según termine correcta o incorrectamente la transacción. Es posible definir transacciones que incluyan a sí mismo transacciones y especificar diversos grados de dependencia entre las mismas.

  A nivel de desarrollo, una transacción implica dos sentencias de código que marcan el bloque *begin-end* de la transacción y la definición como transaccionales de aquellos objetos cuyo estado deba ser actualizado de forma transaccional.

  El soporte transaccional de Forté se complementa con los mecanismos de objetos compartidos y de excepciones. En Forté es posible definir un objeto particular como compartido, *shared*. Mediante esta propiedad Forté sabe que debe encolar las llamadas a este objeto y que las operaciones que realiza el objeto han de serializarse a fin de no perder su integridad. Las excepciones permiten la captura por las aplicaciones de eventos exteriores a las mismas pero que inciden en su ejecución, por ejemplo el llenado de una unidad de disco. Dentro del flujo de la aplicación, las excepciones tienen su fujo propio y Forté salta a él ante cualquier evento que impide la normal ejecución de la aplicación.

- *Tolerancia a Fallos.* La tolerancia a fallos ha pasado de ser algo esotérico, que muy pocos necesitaban y menos podían pagar, a ser un requisito de negocio cada vez más exigido, como permitir por ejemplo com-

Figura 4. Los servicios para Internet de Forté.



prar a cualquier hora en Internet, consultar en cualquier momento el estado de nuestro envío por mensajería o servir pedidos durante 24 horas al día todos los días de la semana.

Tener tolerancia a fallos en aplicación significaba hardware específico, sistemas operativos especiales, codificación de bajo nivel. Forté sitúa la tolerancia a fallos fuera del código, sin impactar al desarrollo. El momento de la instalación de la aplicación es cuando definimos si es tolerante a fallos o no. Para ello se especifica el nodo principal dónde se vas a ejecutar y el nodo de respaldo; y se instala en ambos. Cuando Forté detecta que el nodo principal ha caído, levanta la aplicación en el nodo de respaldo. Ambos nodos no tienen por qué pertenecer a la misma plataforma, por ejemplo un Unix puede dar respaldo a varias máquinas NT o viceversa. En este momento existen sistemas desarrollados en Forté con disponibilidades superiores al 99,9999 %, por ejemplo el sistema de emergencia 911 de la ciudad de Nueva York.

- *Replicación y Balanceo de la Carga.* Escalar un sistema distribuido en el nivel de los clientes es sencillo, añado estaciones de trabajo según se incorporan nuevos usuarios al sistema y ya lo tengo. En el lado de los servidores la situación es algo más complicada, no basta con añadir servidores o incrementar el tamaño de los mismos; pueden existir servidores que sean verdaderos cuellos de botella.

Forté permite utilizar con los servidores la misma estrategia que con los clientes, para ello soporta la réplica de servicios, es decir, mantener activas múltiples copias del mismo servicio en uno o varios servidores. Forté se encarga de repartir equitativamente las peticiones entre todas las copias del servicio. La definición del número de copias activas y dónde se ejecutan puede hacerse tanto en tiempo de instalación como durante la propia ejecución de la aplicación, sin detenerla y sin afectar al código fuente.

Forté permite que diferentes servidores, basados en diferentes plataformas (NT, Unix, VMS, etc.) colaboren soportando los mismos servicios en la misma aplicación. La posibilidad de variar el número de copias activas de cada servicio de forma dinámica es una característica especialmente indicada para aplicaciones Internet, en las que el número de posibles usuarios es potencialmente elevado e imprevisible en su comportamiento.

La arquitectura de Forté posee otras numerosas características, no obstante os remitimos a las referencias al final del artículo para que obtengáis información más en detalle:

- Servicios compartidos entre aplicaciones
- Particionamiento estático y dinámico
- Aplicaciones compiladas
- Plataformas heterogéneas (14 plataformas soportadas)
- Internacionalización de aplicaciones
- Chequeo de Versiones en ejecución
- Portabilidad de la GUI

Integración con otros sistemas

El capítulo de integración es de singular importancia en una herramienta que pretende tener carácter corporativo. La informática de cualquier gran organización es el resultado de sucesivas "olas tecnológicas" cada una de las cuales ha dejado su poso particualr en forma de sistemas. Sistemas que cumplen cada uno un cometido específico y no pueden cambiarse de la noche a la mañana. Por otra parte el mercado nos ofrece aplicaciones ofimáticas, paquetes, y un creciente número de "componentes" que pueden ahorrar un esfuerzo importante de desarrollo a la vez que ofrecen un funcionalidad potente.

Forté dispone de un amplio abanico de interfaces para la conexión de sus aplicaciones con las mencionadas anteriormente. Para revisarlas brevemente las hemos dividido en tres áreas, mundo Microsoft, sistemas abiertos y *mainfame*.

- **El mundo Microsoft.** Microsoft tiene su arquitectura propia para sistemas distribuidos, se llama DCOM y su interfaz más conocida es OLE. Una aplicación Forté puede

comportarse bien como cliente OLE, utilizando los servicios de un servidor OLE como pueda ser Excel, bien como servidor OLE. Como servidor OLE permite acceder a sus servicios desde aplicaciones Visual Basic, macros de Word, etc. El acceso a datos vía ODBC está soportado como si se tratase de un gestor de base de datos más entre los contemplados en el entorno. La tecnología de componentes de Microsoft, ActiveX, también está integrada en Forté. Es posible integrar este tipo de componentes en la interfaz de usuario de aplicaciones Forté.

- **Los sistemas abiertos.** Entendiendo por sistemas abiertos a aquellos que ofrecen interfaces estandarizadas por organizaciones internacionales, Forté ofrece un abanico amplio de interfaces de conexión: RPC, *sockets*, DCE, CORBA 2.0, Encina,Tuxedo. También es posible definir *"wrappers"* o envoltorios de librerías escritas en lenguajes de 3ª generación como C o C++. En estos casos es conveniente asegurar que las librerías gestionan su propia memoria y soportan las hebras (*threads*). Todas las interfaces mencionadas sirven tanto para realizar llamadas desde aplicaciones Forté a otras aplicaciones como al revés.

Existen en el mercado librerías de terceros que integran Forté con otros *middleware* del mercado como MQSeries o EDA/SQL.

- **El mainframe.** Considerando el mundo de los sistemas centrales IBM y compatibles, Forté permite acceder a recursos en este tipo de sistemas de varias formas. Para el acceso a aplicaciones vía su interfaz de usuario, Forté dispone de una librería capaz de capturar pantallas 3270 y extraer o introducir datos en dichas pantallas sin requerir programación específica. Este método no supone cambio alguno para las aplicaciones o la infraestructura existente. Forté se conecta a CICS y puede arrancar transacciones en este entorno, con las que intercambia datos. De esta forma pueden reutilizarse las transacciones existentes y se garantiza la integridad en el acceso a los datos. El acceso directo a los datos en DB/2 puede realizarse a través de las múltiples pasarelas (*gateways*) existentes en el mercado, por ejemplo DRDA de IBM, MDI de Sybase, Oracle.

La plataforma S/390 estará soportada a nivel nativo por Forté en Junio de 1998 de acuerdo a los planes de desarrollo de la compañía.

Estrategia de Forté en Internet

La estrategia de Forté para el desarrollo de aplicaciones Internet pasa por tres fases:

Fase I. En esta fase, correspondiente a la versión actual del producto (R3), es posible acceder a servicios de aplicación Forté desde navegadores con páginas HTML o con programas Java. Para soportar el acceso HTML Forté dispone de conexión con servidores Web siguiendo el estándar CGI. Las páginas Web las puede generar Forté dinámicamente a partir de la GUI o pueden diseñarse a medida, incorporando de este modo sentencias JavaScript. El cliente Java accede a los servicios de Forté a través del protocolo IIOP, soportado por navegadores y servidores Web como Netscape (que incluyen el ORB de Visigenic). Forté soporta los conceptos de applet y de servplet implementados en el lenguaje de Forté.

Fase II. A partir de la GUI, Forté genera código Java que implementa el cliente remoto. Este cliente se envía a la estación cliente remota usando el navegador y accede a los servicios Forté utilizando el protocolo IIOP. En este momento el generador Java se encuentra en versión Beta, a la espera de su liberación definitiva.

Fase III. Esta fase corresponde a la próxima versión del producto (R4) y se caracteriza por la integración total de Java en el entorno de aplicaciones Forté. Java será un lenguaje de programación de Forté, conjuntamente con el 4GL de Forté, TOOL. Desde Java será posible acceder a todos los recursos de la arquitectura Forté. Se programará indistintamente en Java y en Forté en la misma aplicación.

Conclusiones

En Forté encontramos no un entorno de desarrollo al estilo usual, sino uno que abarca tanto el desarrollo, como la instalación como la propia gestión de las aplicaciones en producción. Su implantación requiere por tanto la participación del departamento de informática al completo: desarrollo, sistemas, explotación. Su carácter de herramienta corporativa hace que su adopción sea una decisión estratégica para una empresa y hay que busca en su capacidad el potencial para diferenciar nuestra empresa de la competencia. Tecnológicamente Forté adopta el paradigma de objetos distribuidos como base para su lenguaje 4GL y su entorno de aplicaciones. La fuerte integración de herramientas que proporciona Forté permite rentabilizar el esfuerzo de aprendizaje del entorno y apunta hacia una alta productividad. Su capacidad multiplataforma e integradora de sistemas le hace ser una pieza esencial para organizaciones con una base informática instalada con múltiples sistemas heterogéneos y una fuerte necesidad de procesos de negocio integrados. En Internet Forté aporta una sólida plataforma servidora, con tolerancia a fallos y escalabilidad elevada capaz de atender soluciones críticas como comercio electrónico o centros de servicio.

PROGRAMACIÓN DE JUEGOS

Generador de movimientos

Eugenio Castillo Jimenez

Los movimientos (incluyendo el movimiento nulo) constituyen los nodos del árbol de posibilidades que un programa puede llegar a generar, a su vez el resultado de un análisis se da en la forma de una secuencia de movimientos. El modo en que estos se generan, almacenan y ordenan constituye pues una parte fundamental en un programa de ajedrez.

Para representar el tablero y sus piezas utilizaremos las estructuras de datos referidas en artículos anteriores. Veamos cómo se definieron:

```
type { Casillas: a1=0, b1=1,..., a2=$10,..., h8=$77
      }
TipoCasilla  =   0..$77;  { Las 64 casillas, aunque
      son declaradas 120}
TipoColor   = (Blancas, Negras);
TipoPieza   =
      (Vacio,Peon,Caballo,Alfil,Torre,Dama,Rey);
TipoIndex  = 0..15;      { Indexado sobre
      TabPiezas }
TableroPiezas : array[TipoCasilla] of TipoPieza;
      {Array con los tipos de piezas por casilla}
TableroColor : array[TipoCasilla] of TipoColor;
      {Bando de la pieza sobre la casilla}
TableroIndex : array[TipoCasilla] of TipoIndex;
      {Indice sobre el array de piezas por bando}
```

Para definir la estructura de un movimiento en primer lugar hemos de conocer los elementos que lo componen:

1. Pieza que mueve.
2. Casilla origen.
3. Casilla destino.
4. Si es un movimiento especial:
   a) se trata de una coronación, tipo de coronación.
   b) captura.
   c) captura al paso.
   d) enroque.
5. Posible valor de la variante.
6. Tipo de generador original o tipo de movimiento.
7. Pieza capturada.

Algunos de estos datos son optativos, de hecho se puede representar una jugada con sólo 3 bytes, origen, destino y la especificación del movimiento especial únicamente referido a las coronaciones. Veamos ahora la utilidad de los otros datos incluidos:

Pieza que mueve. Puesto que ya conocemos la casilla original de la pieza nos basta con leer en el array del TableroPiezas para conocer el tipo e indexado de la pieza. Sin embargo se dan casos ante todo de naturaleza informativa (por ejemplo cuando vamos a mostrar una línea principal de resultado) en el que la pieza que mueve no necesariamente se encuentra en la casilla prevista puesto que nos estamos refiriendo a posiciones posteriores. Otro tanto cabe decir de los movimientos especiales como capturas al paso y enroque.

Posible valor de la variante. En algunos programas se usa como criterio de ordenamiento de los movimientos.

Tipo de generador original o tipo de movimiento. Algunos movimientos han sido generados con un fin previo o han surgido provocados por determinadas situaciones, entre estos casos podemos encontrar las recapturas, los movimientos killer, etc. El especificar el tipo u origen nos puede ser de gran utilidad a la hora de depurar un programa o para buscar contrajugadas al tema en curso.

Pieza capturada. Se insertan datos (tipo de pieza o indexado) de la pieza capturada.

Así pues ya podemos definir una estructura de datos válida para nuestras necesidades, veamos un ejemplos:

```
type
tmov = record
        Pieza : TipoPieza;
        Origen,Destino : TipoCasilla;
        Especial : byte;
        Valor : integer;
        TipMov :
    (Normal,Killer,Recaptura,Jaque...);
        PiezaCapturada : TipoPieza;
      end;
const
Captura     =  16;
ApCaptura  =  32; {Captura al paso}
Enroque    =  64;
Coronación = 128;

{Valores de ejemplo para movimientos:

 Cg1-f3 (Caballo tres alfil rey):
  (Pieza := Caballo;Origen := 6;Destino :=
      21;Especial := 0;Valor := 0;TipMov :=
      Normal;PiezaCapturada := Vacio);
  Pe4xd5 (Peón de e4 por peón de d5):
 (Pieza := Peon;Origen := 28;Destino :=
      35;Especial := Captura;Valor := 100 -
      Peon;TipMov := Normal;PiezaCapturada :=
      Peon);
 Pe7-d8=D (Peón de e7 a e8  coronando Dama)
 (Pieza := Peon;Origen := 52;Destino :=
      60;Especial := Coronación+Dama;Valor :=
      800;TipMov := Normal;PiezaCapturada :=
      Vacio);
 Re1-g1 (Enroque corto)
 (Pieza := Rey;Origen := 4;Destino := 6;Especial
      := Enroque;Valor := 25;TipMov :=
      Normal;PiezaCapturada := Vacio);
}
```

Obsérvese en los ejemplos las siguientes cuestiones:

1. En Pe4xd5 la variable **valor** es (100 - Peon), aquí Peon posee el valor 1, así el resultado final sería 99, en caso de que capturasemos con otra pieza como una Torre la expresión sería (100 - Torre) que es 96. La finalidad de esta operación consiste en atribuirle un valor más alto a la captura cuanto menor sea el valor de la pieza que toma, de esta forma al ordenar los movimientos de mayor a menor en caso de que más de una pieza pudiera capturar a otra se comenzaría por aquella que fuera de valor más bajo.

2. La jugada de coronación Pe7-d8=D inserta en la variable Especial el valor (Coronación + Dama), en realidad esta expresión es equivalente a (Coronación or Dama) puesto que los bites de ambos valores no llegan a coincidir (una operación "and" daría 0), de esta forma expresamos en un byte que se trata de un movimiento de coronación y el tipo de coronación que se trata. Observese que las constantes Captura..Coronación son potencias de 2 y comienzan a partir del valor 16 precisamente para permitir lo comentado con anterioridad (los valores Peon..Rey van del 1 al 6).

Tanto el generador de movimientos como el sistema de evaluación son los dos algoritmos que consumen mas tiempo, por lo tanto el enfocarlos para alcanzar la máxima velocidad posible es algo fundamental, estudiemos los distintos tipos de movimientos a tratar y sus posibles desarrollos.

En primer lugar podemos encontrar las piezas que siguen movimientos rectilíneos (el GNU los denomina piezas "sweep"), ya sean diagonales, columnas o líneas, aquí podemos utilizar vectores (incrementos de posición) para calcular el movimiento. La particularidad de estas piezas reside en que apenas "topan" con una casilla ocupada o un borde detienen su movimiento, por tanto a cada incremento que se produzca habremos de efectuar dos preguntas: primero saber si estamos fuera del tablero y segundo ver si hay alguna pieza, para este último caso si la pieza de la casilla es de color contrario generamos un movimiento de captura.

```
procedure movpiez (sq,              {posición de la
        pieza}
              tipo,     {tipo de pieza}
              contrario : integer)
var
pos,f : integer;
dirección : integer;
label nuevadir;
  . .
begin
 for f := 1 to numdir [tipo] do {numdir contiene el
      número de direcciónes, torre y alfil 4, dama 8}
 begin
   pos := sq; dirección = dir [tipo,f]; {dir indica la
      pendiente}
   while (pos and 77) = 0 do {Si "pos and 77" es
      distinto de cero significa que estamos fuera
      del tablero}
   begin
     pos := pos + dirección;
     if TableroPiezas [pos] <> 0 then
     begin
       if TableroColor [pos] = contrario then
         InsertaCaptura (pos,sq);
       goto nuevadir:
     end
     else
       InsertaMNormal (pos,sq);
   end;

nuevadir:

 end;
end;
```

Este sistema mostrado es muy parecido para los movimientos de rey, caballo, y peón, sólo que más simple puesto que carece del proceso de incremento de pos dentro del bucle while. Una buena parte de los programas de ajedrez usan este método aunque de un modo más sofisticado evitando las llamadas a subrutinas y depurando el código en ensamblador.

El GNU utiliza una idea diferente que consiste en generar previamente todos los movimientos posibles para cada tipo de pieza (y bando) desde cada posición del tablero (los tableros GNU son de 64 bites y no de $77), el consumo de memoria que ello requiere no es excesivo (64x64x6x2 = 49152 bytes). Con este sistema nos ahorramos por de pronto las preguntas referidas al chequeo de los bordes del tablero, la inicialización del incremento por dirección y el incremento en sí, como contrapartida hemos de chequear tras cada iteración para saber si hemos llegado al final y el incremento tras cada movimiento es sustituido por la búsqueda de la siguiente casilla, por lo que el ahorro de tiempo es mínimo.

```
struct sqdata {
  short nextpos;
  short nextdir;
};

struct sqdata posdata[8][64][64];

/* posdata[tipopieza][casillaorigen][casilladestino]
ejemplo;
```

El primer movimiento para un alfil situado en e8 sería posdata [alfil][e8][e8].nextpos, si en f7 hubiera una pieza bloqueando la posición posdata [alfil][e8][e7].nextdir nos daría la casilla inicial de la siguiente dirección. En caso de que nos topásemos con un borde tanto nextpos como nextdir nos darían la casilla válida, de hecho aquí no es necesaria esta comprobación. El final de la generación se da cuando la casilla destino es la misma que el origen.

```
p : puntero de posdata;
u : variable casilla;
sq : casilla original de partida;
*/

  p = posdata[piece][sq];
  u = p[sq].nextpos;
  do {
   if (color[u] == neutral) {
     LinkMove(ply,sq,u); //LinkMove inserta el
     movimiento en una buffer
     u = p[u].nextpos;
   }
   else {
     if (color[u] == xside) LinkCapture(ply,sq,u);
     u = p[u].nextdir; //hemos topado con una
     pieza y es preciso tomar otra dirección
   }
  } while (u != sq);
```

El CRAFTY (y otros programas como el DARK THOUGHT alemán) utiliza un sistema ideal para procesadores de 64 bits, que consiste en generar los movimientos a partir de mapas de 64 bits que representan todo el tablero (aprovechando la coincidencia). Representemos los movimientos de una torre situada en e4:

```
{ 0, 0, 0, 0, 1, 0, 0, 0,
  0, 0, 0, 0, 1, 0, 0, 0,
  0, 0, 0, 0, 1, 0, 0, 0,
  0, 0, 0, 0, 1, 0, 0, 0,
  1, 1, 1, 1, 0, 1, 1, 1,
  0, 0, 0, 0, 1, 0, 0, 0,
  0, 0, 0, 0, 1, 0, 0, 0,
  0, 0, 0, 0, 1, 0, 0, 0};
```

El procesador del CRAY donde Robert Hyat ejecutaba las versiones iniciales del CRAFTY era capaz de identificar con una sola instrucción el bit más alto de una palabra de 64 bits, con lo cual vería altamente facilitada la labor de ir desgranando las posibles casillas a las que pudiera moverse una pieza. La simplicidad del programa quedaría representada así:

```
void (dlword sq,masc)
{ int pos;

  while (masc)
  { pos = siguiente (masc);
    if (TableroPieza [pos])
    { if (TableroColor [pos]) //comprobación si es
      del color contrario
        LinkaCaptura (sq,pos);
    }
    else
    { LinkaMov (sq,pos);
    }
  }
}
```

La rutina "siguiente" devuelve la posición del bit encendido más alto de **masc** y lo desactiva para no volverlo a leer, evidentemente al final **masc** tendrá el valor 0 y provocará el fin del bucle while. Como podemos ver este sistema parece ser el más simple y rápido, siempre que encontremos un algoritmo suficientemente eficaz para los ordenadores compatibles PC en los que no existe ninguna instrucción milagrosa que resuelva la papeleta del "bit más alto".

Los programadores del DARK-THOUGH, E.A. Heinz y M. Gille de la universidad Alemana de Karlsruhe también se encuentran ante el mismo problema y dan el siguiente algoritmo desarrollado a partir de referencias del libro "Hacker's Memory":

```
#define m1 ((unsigned_64)0x5555555555555555)
#define m2 ((unsigned_64)0x3333333333333333)

unsigned int siguiente (const unsigned_64 b) {

  unsigned_32 n;
  const unsigned_64 a = b - ((b >> 1) & m1);
  const unsigned_64 c = (a & m2) + ((a >> 2) &
     m2);
  n = ((unsigned_32) c) + (c >> 32);
  n = (n & 0x0f0f0f0f) + ((n >> 4) & 0x0f0f0f0f);
  n = (n & 0xffff) + (n >> 16);
  n = (n & 0xff) + (n >> 8);
  return n;
}
```

Según sus autores este código se ve reducido a 15 intrucciones de procesador despues del proceso de compilación, lo cual es excesivo para obtener velocidades razonables.Tambien se puede recurrir a dividir los 64 bits en 4 grupos de 16 bits y rellenar un espacio de 64K con todas las posibilidades que da un entero conociendo para cada caso cual es su bit más alto (o recurrir a búsquedas dicotómicas), de cualquier modo no parece existir una combinación menor de 4 de intrucciones (el máximo aceptable para adoptar este método) capaz de dar una solución.

Sin embargo la idea del mapa (o máscara según autores) de bits es muy interesante y tal vez la solución se encuentre en emplear mapas más pequeños. Por ejemplo el área cubierto por una torre no puede exceder de 15 casilla incluyendo la suya propia, otro tanto ocurriría con un alfil (en este caso el área es aún inferior), perfectamente se podría reducir el área de interés a un entero de 16 bits.

Veamos previamente cómo se extraen e insertan datos de los mapas originalmente diseñados por Robert Hyat. Su idea inicial consistía en crear 4 tableros representando los dos primeros las 8 columnas y 8 lineas de los movimientos de torres y damas, los dos últimos representaban las dos diagonales posibles para los alfiles y damas (existen más de 8 diagonales pero ninguna de más de 8 escaques), como se puede constatar todos están formados por conjuntos de 8 bits.

Cada pieza del tablero activaba un bit en los TablerosMap (tableros de ma-

pas) en su correspondiente columna, línea y diagonales. De este modo obtenemos una imagen del tablero que sólo tiene en cuenta si las casillas (representadas por bits) estan ocupadas o vacias.

Supongamos que la columna B (la segunda columna del tablero) tiene este aspecto:

A) (1 0 0 1 0 1 0 1)
NBit 1 2 3 4 5 6 7 8

cada bit encendido representa una pieza, digamos que una torre se encuentra en el bit 4 (posición b4 sobre el tablero) sus posibles movimientos incluyendo capturas serían :

B) (1 1 1 0 1 1 0 0)

incluyendo sólo capturas tenemos:

C) (1 0 0 0 0 1 0 0)

e incluyendo sólo movimientos tenemos:

D) (0 1 1 0 1 0 0 0)

Otro tanto sucedería con su movimiento sobre las líneas. Podemos ver que para cada columna (línea, y diagonales) existen 256 casos posibles y para cada uno de estos casos se dan 8 posiciones relativas para la pieza que mueve (en realidad no es completamente cierto pero el resultado final no induce a error y se mantiene constante la longitud de lo arrays a declarar). Por tanto tenemos 8 columnas por 8 posiciones relativas por 256 posibilidades = 16384 casos posibles de diferentes movimientos del tipo (B) y cada uno ocupa un byte.

Ahora la rutina "siguiente" se podría realizar dentro de unos márgenes razonables de velocidad, pero aún podemos ir más allá, si retocamos el proceso generador del siguiente modo:

```
#define siguiente(masc,pos){\
 pos = nextbitpos [masc];\
 masc ^= xormasc [masc];\
}
/*nextbitpos nos dice la posición del bit más alto,
     en realidad da directamente la posición de la
     casilla ahorrando tiempo,
  xormasc da el valor necesario para desactivar
     mediante una operación xor el primer bit ac-
     tivo de masc.*/
void (word sq)
{ int pos;
 //1° estudio de columnas
 masc = TableroMapCapturas [columna [sq]]; //
     mapa bits (C)
  while (masc)
  { siguiente (masc,pos);
    if (TableroColor [pos])
      LinkaCaptura (sq,pos);
  }
 masc = TableroMapMovs [columna [sq]]; // mapa
     bits (D)
 while (masc)
 { siguiente (masc,pos);
   if (TableroColor [pos])
     LinkaMov (sq,pos);
 }
//2° Estudio de las líneas
..........................
}
```

Como podemos deducir nos hemos ahorrado una comparación "if (TableroPieza [pos])" a diferencia del anterior código mostrado puesto que ya sabemos con antelación si habrá o no pieza sobre la casilla a la que vamos. Es cierto que como contrapartida ahora hemos de inicializar **masc** con "masc = TableroMapMovs [columna [sq]]" pero apenas la pieza genere más de dos movimientos válidos comenzamos a ganar velocidad.

Tras casi 40 años de historia en desarrollos de algoritmos para generar movimientos a "toda pastilla" he de mostrar otro método realmente simple y efectivo que produce unos ahorros parecidos al anterior sin emplear ninguna clase de información accesoria del tipo mapbits o los arrays de pendientes, etc.

```
void movtorre (word sq) //movimiento de las torres

{
 //pendiente +1
 if (TableroPieza [sq + 1])
 { if (TableroColor [sq + 1])
     LinkaCaptura (sq,sq + 1);
     goto pendientem1;
 }
 else
   Linkamov (sq,sq + 1);
 if (TableroPieza [sq + 2])
 { if (TableroColor [sq + 2])
     LinkaCaptura (sq,sq + 2);
   goto pendientem1;
 }
 else
   Linkamov (sq,sq + 2);
 if (TableroPieza [sq + 3])
 { if (TableroColor [sq + 3])
     LinkaCaptura (sq,sq + 3);
   goto pendientem1;
 }
 else
   Linkamov (sq,sq + 3);
 ........//hasta sq + 7

 pendientem1:
 //pendiente -1
 if (TableroPieza [sq - 1])
 { if (TableroColor [sq - 1])
     LinkaCaptura (sq,sq - 1);
   goto pendiente8;
 }
 else
   Linkamov (sq,sq - 1);
 if (TableroPieza [sq - 2])
 { if (TableroColor [sq - 2])
     LinkaCaptura (sq,sq - 2);
   goto pendiente8;
 }
 else
   Linkamov (sq,sq - 2);
 ........//hasta -7
 pendiente8:

 //pendiente +8
}
```

Este nuevo sistema aparatoso en un principio, pero efectivo, se basa en el principio del "despilfarro del código" lo que anteriormente hemos representado mediante bucles (ahorrando código) ahora lo hacemos a pelo (despilfarrando) bucle a bucle, basandonos en que el número de bucles por pendiente jamás superará las 7 iteraciones (de otro modo sería excesivamente tedioso o imposible), con lo que entre movimiento y movimiento nos ahorramos la comparación para ver si hemos finalizado el bucle, además del salto al principio del mismo.

También la comparación "if (TableroPieza [sq - 2])" produce un ahorro de código puesto que el compilador evita el cálculo de "- 2" (valor de la pendiente) dado que lo resta del offset de "TableroPieza".

En todos estos ejemplos vistos hasta ahora no nos hemos involucrado en el contenido de "Linkamov (sq,sq - 2);" que introduce el movimiento en una lista. Esta operación depende en gran medida del criterio que escojamos para ordenar los movimientos y del modo en que estos se vayan a ejecutar. Existe otra "generación" de programas que no introducen los movimientos en una lista, sino que los realizan a medida que los genera. De este modo si una jugada nos dá un salto por beta (jugada suficientemente buena como para no seguir analizando, equivalente a una refutación) no es preciso generar el resto de las jugadas del turno en cuestión, produciendose de este modo un considerable ahorro de tiempo siempre que los saltos por beta sean suficientemente numerosos, en caso contrario puede llegar a ser incluso perjudicial.

Orden de movimientos

Cuanto antes analice un ordenador la serie de jugadas de la variante principal, mayor será el ahorro de nodos en el árbol generado, las diferencias entre considerar esto y no considerarlo pueden suponer un gasto extra de un 200% de media en el tiempo de respuesta para obtener los mismo resultados. Es evidente pues que el orden de los movimientos es muy importante.

Esta es la razón primordial por la cual el ordenador recurre a búsquedas iterativas con incrementos de profundidad. Cada iteración del nivel anterior le da una variante principal, la cual es aprovechada para iniciar con ella la siguiente iteración.

Los primeros estudios realizados revelaron que si se ordenaban los movimientos según el valor de las piezas que mueven de menor a mayor, esto es, primero los peones y luego el rey, se producía un cierto incremento en el ahorro de nodos. Posteriormente se descubrió que si se realizan primero las capturas, aunque estas no sean rentables sino perjudiciales, el ahorro se incrementaba. Dentro de las capturas si estas se ordenan poniendo en primer lugar a las capturas de mayor peso, esto es de dama a peón (de mayor a menor valor) se produce otra mejora, a su vez dentro de este suborden conviene tomar primero con las piezas de menor peso, o sea de nuevo de menor a mayor.

Esquema de preferencias:

1. Variante Principal.
   1.1. Si no existe Variante principal ver si hay algo en las tablas hash.
2. Capturas.
   2.1. Captura de la última pieza.
   2.2. Resto de capturas ordenadas según el peso de la pieza capturada (mayor a menor).
   2.2.1. Ordenar por el peso de la pieza que toma (menor a mayor).
3. Movimientos Killer.
4. Killers "históricos".
5. Enroque.
6. Resto de movimientos.

Como podemos ver incorporamos nuevos conceptos como "tablas hash" y "Killers". Para estudiar más en detalle las primeras veamos su estructura:

```
struct hashentry
{ unsigned long hashbd;
  unsigned short mv;
  unsigned char flags, depth,color;
  short score;
};
```

hashbd: es una clave encriptada de la posición del tablero, mv : contiene el movimiento dado como mejor en a posición dada, flags: información accesoria del movimiento, depth: profundidad que restaba por analizar, color: bando que mueve, score: valor que obtuvo la respuesta. Cuando el programa no encuentra una jugada de variante principal recurre a las tablas hash para saber si la posición se dió con anterioridad y trata de analizar en primer lugar la jugada dada por buena en la ocasión precedente. **depth** nos dice la profundidad que restaba por analizar, lo cual nos revela la potencia del análisis. Por ejemplo si sabemos que en la posición X el movimiento g1-g3 fué el mejor restando 4 jugadas de profundad de análisis, la fiabilidad de la respuesta será mejor que cuando restan 3, puesto que en el primer caso se han analizado más jugadas. Este hecho se aprovecha para realizar a veces "trasposiciones", es decir, que ya se da por buena la jugada de la tabla hash devolviendo en el nodo en cuestión el valor **score** del hash, puesto que la posición ya fue analizada con anterioridad a una profundidad igual o mayor que la actual.

Hay que hacer constar que se han de incluir los valores alfa y beta en curso en el hash puesto que puede influir en el resultado, y pueden hacer inválida una trasposición.

Los movimientos **Killer** son aquellos que en el nodo inmediatamente inferior (la jugada precedente del contrario) fueron considerados como mejores jugadas para el bando en curso. Es de cierta lógica el suponer que en una buena parte de casos la mejor jugada tienda a ser la misma en situaciones parecidas, en algunos casos es algo completamente falso, pero a nivel estadístico funciona.

Los **Killer Históricos** son parecidos a sus hermanos Killer sólo que poseen un valor menor dado que no se producen en situaciones inmediatamente precedentes sino que a lo largo de todo el árbol. Supongamos que la jugada h2-h4 ha sido la mejor en 150 ocasiones y h2-h3 lo ha sido en sólo 20, así pues el Killer Histórico se encarga de recoger esta información y añadir un plus al valor de ordenamiento de las jugadas. El efecto estadístico de este concepto ha demostrado su efectividad.

Estos son algunos lugares interesantes de internet para obtener información a nivel de programación aplicada al ajedrez:

- www.gambitsoft.com
- www.dcs.qmw.ac.uk/~icca
- www.chess-space.com

Conexión a Internet a través de Linux

Antonio José Novillo López

En este artículo vamos a abordar la posibilidad de conectar nuestro equipo o estación de trabajo con una cuenta exterior a través de una conexión PPP.

El funcionamiento de la red será independiente del método que usemos para conectarnos, ya sea una tarjeta Ethernet a través de una red local, una conexión PPP o una SLIP. De hecho, gran parte de las opciones de configuración de red serán idénticas para los distintos sistemas de conexión, variando tan sólo algunas opciones y demonios que deberemos ejecutar. Aquí nos hemos decidido por una conexión PPP pues la más habitual siempre que nos conectemos desde nuestra casa. Las conexiones Ethernet son más propias de una Universidad o de una red local con un router como conexión al exterior.

Para conseguir nuestro objetivo necesitaremos:

- Un PC con Linux instalado. Cualquier distribución nos será valida, aunque variarán los ficheros de configuración. Será preferible disponer un kernel 2.0.x. con soporte para Red compilado.
- Software de Red. Las tres grandes distribuciones (Slackware, Debian y Red Hat) contienen opciones para instalar este software durante la instalación del sistema. En caso de tener instalado el sistema sin estas opciones, siempre se pueden instalar a posteriori.
- Un módem. Posteriormente se describirán con más detalle los módems válidos en Linux.

Configuración del módem

Algunas distribuciones, caso de Slackware, contienen scripts que configuran directamente el módem a través de una serie de preguntas al usuario. Nosotros nos pondremos en el peor caso y realizaremos la instalación de éste directamente.

En primer lugar, debemos asegurarnos que nuestro módem está soportado por Linux. Este sistema soporta tanto módems internos como externos, siendo estos segundos preferibles salvo en el caso de que no dispongamos de UARTs rápidas. Debemos evitar los winmódems, pues no poseen chip inteligente y realizan sus funciones lógicas a través de software, que, habitualmente, no está y no estará disponible para el sistema operativo Linux. Tampoco son recomendables los módems PnP, que aunque se puede trabajar con ellos, son bastante problemáticos a la hora de realizar la configuración. Para obtener más información sobre cómo uti-

Hoy en día la posibilidad de conexión a redes es un opción fundamental dentro de un Sistema Operativo. La potencia de la integración de la funciones de red dentro del kernel y la versatilidad de este sistema han convertido a Linux en una opción más que interesante para acceder a las grandes redes como Internet o la española Infovía.

lizar este tipo de módems, ver el Serial-HOWTO.

Un problema habitual que suele ocurrir al configurar el módem, consiste en intentar que Linux comparta interrupciones IRQ, algo que el sistema no soporta. Este problema se da especialmente en los módems internos, que suelen usar una configuración DOS COM4, IRQ 3. Si es éste nuestro caso, dispondremos de varias posibilidades distintas para solventar el problema:

- Configurar el módem para que utilice otro puerto, como podría ser el COM2, /dev/ttyS1 en Linux.
- Cambiar la IRQ que utiliza el puerto interno del módem.
- Desconectar el puerto COM2 de la placa (serie o madre) o modificar la IRQ que utiliza.

Seguidamente deberemos configurar los dispositivos serie. Para ello deberemos editar el fichero /etc/rc.serial en distribuciones Slackware y el /etc/rc.d/rc.serial en distribuciones Red Hat. Este fichero se ejecuta al arrancar el sistema y configura los distintos puertos serie. Para modificarlo deberemos trabajar como usuario root, pues en caso contrario no dispondremos de permisos de escritura sobre el fichero. La sentencia genérica a incluir será:

```
# setserial –v /dev/ttySX irq N <spd_ vhi>
```

Donde X es el número del puerto (0 para COM1, 1 para COM2, 2 para COM3 y 3 para COM4) y N es el número de IRQ asociado a dicho puerto. Para ver con qué dispositivos están configuradas cada IRQ puede ejecutar:

```
$cat /proc/interrupts.
```

El parámetro *spd_vhi* se deberá incluir cuando nuestro módem sea superior a 22.8 Kbps. En el caso de disponer de varios módems, deberemos incluir tantas líneas, como dispositivos tengamos conectados. Se puede crear un link simbólico como dispositivo /dev/modem del puerto serie en el que se encuentre instalado nuestro módem:

```
# ln –s /dev/ttySX /dev/modem
```

Para más información sobre el comando setserial, Serial-HOWTO y/o la página de man del comando.

Configuración del Kernel

Para poder realizar una conexión PPP necesitaremos que nuestro kernel tenga soporte PPP. Esto se puede conseguir de dos maneras diferentes. Por una parte, recompilando el kernel del sistema con la opción de soporte PPP. De este modo, nuestro kernel tendrá siempre cargado dicho soporte.

El problema es que, si vamos a efectuar conexiones puntuales no muy prolongadas, el tamaño del kernel es mayor y no está utilizando habitualmente las funciones de PPP. Por ello, una solución más óptima desde el punto de vista de memoria, consiste en compilar el soporte PPP como módulo. Esta opción nos generará los módulos shlc.o y ppp.o, que será necesario cargar en memoria para que el sistema soporte la conexión adecuadamente. Está se realiza mediante:

```
#modprobe ppp.
```

No deberemos cargar el módulo hasta que vayamos a realizar la instalación y deberemos ser el usuario root para poder realizar esta operación. Otra posibilidad más elegante sería configurar kernel para que cargue el módulo cuando se produzcan las llamadas al núcleo, pero es más complicado y está fuera del objetivo de este artículo. Para más información sobre la compilación del kernel y las opciones de carga y comprobación de los módulos, ver Kernel-HOWTO y Module-HOWTO, así como el el Linux Kernel Hacking Guide.

Protocolo PPP

El PPP es un protocolo utilizado para enviar datagramas a través de una conexión serie, permitiendo a las partes comunicantes negociar opciones como las direcciones IP y el tamaño máximo de los datagramas. En Linux la funcionalidad del PPP está dividida en dos partes. Un controlador HDLC de bajo nivel que se encuentra situado en el kernel (que es el módulo que hemos configurado anteriormente), y un demonio, pppd, que controla los diferentes protocolos de control. También necesitaremos un programa que se encargue de establecer la conexión telefónica con el proveedor de servicios. El programa habitual para realizar esta conexión es *chat*.

El funcionamiento general consistirá en establecer la conexión telefónica con el servidor a través del programa *chat*, para posteriormente pasar la línea a modo PPP a través del comando pppd. A partir de este momento podremos utilizar cualquier protocolo que esté basado en TCP/IP. Las distribuciones normales habitualmente incluyen las herramientas básicas, como son ftp, telnet, sendmail o ping. Mediante estas herramientas podremos conseguir otros programas que estén más orientados al usuario, como son Netscape, Mosaic y otros clientes gráficos para correo y ftp. Cuando deseemos finalizar la sesión, volveremos a usar *chat* para indicarle al módem que finalice la conexión actual.

Una sentencia habitual usando *chat* sería la siguiente:

```
# chat –v ABORT 'NO CARRIER'
ABORT BUSY '' ATZ
OK ATDT$NUMERO CONNECT ''
```

La opción –v sirve para que el comando redireccione los resultados del mismo hacia el fichero indicado por local2 en la configuración de syslog. Para indicar una fichero diferente al por defecto, basta con introducir la siguiente línea en el fichero /etc/syslog.conf, donde PATH indica el camino del fichero deseado: local2.*PATH

Lo siguientes son parámetros que mandaremos a través del módem. Los dos primeros, los ABORT, se usan para cortar la comunicación cuando el módem obtiene una señal de línea ocupada o no descuelga. Los comandos incluyen el que nosotros enviamos así como el que esperamos recibir.

En primer lugar se coloca una cadena vacía, indicando que nos da igual la señal con la que nos conteste el módem. ATZ es el comando para inicialización de módems compatibles Hayes. Si no es el caso, se puede eliminar. Nosotros esperamos que nos responda con OK. ATDT marca el número de teléfono NUMERO, que será el del proveedor o bien 055 en el caso de querer conectar con Infovía. Posteriormente esperamos recibir el comando CONNECT que nos indica que la conexión ha sido un éxito y la línea está preparada para pasar a modo PPP. La utilidad *chat* tiene más opciones y posibilidades de las aquí explicadas y se encuentran detalladas en su página de man.

El último paso para poder conectarse a Internet es la ejecución de pppd

Ha llegado el momento de ejecutar pppd, último paso previo para poder introducirnos en la red de redes. Este comando posee también gran número de opciones, aunque explicaremos tan sólo las que nos van a ser útiles. Dado que el número de estas opciones a utilizar puede ser elevado, conviene incluirlas en unos ficheros de opciones. Estos son dos: */etc/ppp/options*, que contiene las opciones generales para todos los usuarios y *.ppprc*, que contiene las opciones específicas de cada usuario. Si en nuestro equipo, como suele ser habitual en un ordenador casero, sólo entra un usuario, se pueden colocar todas las opciones directamente en */etc/ppp/options*. Un ejemplo podría ser:

```
connect /etc/ppp/dial-isp
crtscts
módem
+ua /etc/ppp/password
noipdefault
lock
defaultroute
asyncmap a0000
/dev/módem
38400
mtu 1500
```

Expliquemos un poco el significado de cada línea. La primera indica al programa que *script* vamos a utilizar para conectarnos telefónicamente con el proveedor. El fichero que le indiquemos debe ser ejecutable y contener la línea con el comando *chat* comentada anteriormente. Un ejemplo sería:

```
#!/bin/sh
chat -v " ATDT$NUMERO CONNECT "
```

El parámetro *crtscts* indica que el control de flujo se efectuará por hardware mediante señales CTS (Clear To Send) y RTS (Ready To Send) y no con Xon y Xoff. La palabra módem activa algunas acciones específicas para modem sobre el dispositivo serie, como colgar la línea antes y después de la llamada. La siguiente línea es muy importante, pues indica al demonio que la autenficación PAP se realizará sobre los datos incluidos en el fichero */etc/ppp/password*. El formato de este fichero será el siguiente: login@dominio * password donde login será nuestro identificador en la máquina del proveedor y password la palabra de paso que tengamos asignada.

Dominio será el del proveedor. Para el caso de conectarnos a Infovía, en lugar de login@dominio y password colocaremos en ambos campos infovía. El fichero es un fichero ASCII, luego no es un protocolo seguro, pues cualquiera podría leerlo. Conviene por tanto cambiar los permisos de lectura/escritura de este fichero para que sólo nosotros podamos leerlo. Si no indicamos el nombre del fichero el demonio leerá por defecto */etc/ppp/pap-secrets*.

El siguiente parámetro es *noipdefault*. Habitualmente no dispondremos de una dirección IP fija, sino que el proveedor nos la asignará dinámicamente cuando nos conectemos. Con esta opción le indicamos al demonio que no utilice ninguna IP que encuentre en los fichero de configuración de red, sino que utilice la que le asigne el proveedor. Si nuestra dirección fuera fija, podríamos pasársela como parámetro al demonio, pero este es un caso poco habitual.

Lock establece el método de bloqueo de los puertos UUCP

A continuación encontramos la palabra *lock*. Este parámetro establece el sistema de bloqueo de puertos de UUCP. Cuando estemos utilizando el puerto lo bloqueará para evitar que otros usuarios puedan acceder a él.

Seguidamente encontramos *defaultroute*. Esto indica al demonio que debe introducir en la tabla de ruteo una nueva entrada en la que el dispositivo PPP se coloca como ruteador por defecto. Esto indicará al sistema que todas las direcciones que no se encuentren definidas en la red local, que en nuestro caso no existirá, se encuentran al otro lado de la conexión PPP. El concepto de tabla de ruteo tiene sentido cuando nuestra máquina se encuentra conectada a varias redes locales por diversos dispositivos, pero en nuestro caso no tiene mucho sentido.

El siguiente parámetro, *asyncmap a0000*, previene al demonio de enviar los caracteres ASCII 17 y 19, usados para XON y XOFF. Se podría decir que actúa como un filtro. Si modificamos el número podremos filtrar otros caracteres. Acto seguido indicamos el dispositivo a utilizar, en este caso */dev/modem*, acceso directo creado anteriormente. Por último indicamos la velocidad. Si el valor de su módem es menor de 22800 deberá utilizar otro valor. Por último mtu 1500 indica el tamaño máximo de unidad de transferencia.

Después de ejecutar pppd finalmente se establece la conexión con el proveedor de servicios

Una vez ejecutemos *pppd*, se procederá a la conexión con el proveedor. Una vez ésta se haya conseguido, el demonio pasará a background y quedará a la espera de desconectar. A partir de este momento podremos hacer uso de todos los servicios TCP/IP accesibles en Internet. Cuando deseemos finalizar la conexión bastará con matar el demonio *pppd*. Esto se puede realizar con *# killall pppd*. Otra solución más elegante consiste en incluir una nueva opción el fichero de opciones:

```
disconnect "/usr/sbin/chat –v TIMEOUT 3 "
'\r' '\r' '\+++\c' '\r' ATH0 OK ATH0 OK
```

Para más información sobre la sintaxis y las opciones del comando *pppd* leer la página man y ver la Guía del Administrador de Redes en Linux, traducida al castellano.

Ultimos detalles

Finalmente, deberemos realizar algunas modificaciones en los ficheros de configuración general de red del Sistema. En primer lugar, deberemos editar el fichero */etc/resolv.conf* que contiene información relacionada con el servidor de nombres. Un ejemplo sería

```
domain Nuestro Dominio
nameserver xxx.yyy.zzz.www
```

donde incluiremos el dominio de nuestro proveedor y el servidor de nombres que este posee. El servidor de nombres es la máquina que posee el proveedor donde se encuentra la tabla que relaciona las direcciones IP de cada equipo conectado a Internet con su nombre.

La dirección del servidor de nombre es un dato que nos tiene que ofrecer el proveedor. Podríamos tener varios servidores de nombre, incluyendo tantas líneas con el parámetro nameserver como servidores conozcamos.

Para que funcione el DNS, protocolo utilizado por el servidor de nombres, debemos asegurarnos que el fichero */etc/host.conf* es de la forma:

```
order hosts, bind
multi on
```

Por último, podemos incluir los lugares a los que accedemos más habitualmente en el fichero */etc/hosts*. Éste sería de la forma:

```
#dirección IP nombre_completo nombre

127.0.0.1 loopback nuestra_máquina

0.0.0.0 nuestra_maquina_con_dominio
```

Como vemos la forma general de este programa incluye en cada línea una máquina, de la que daremos su dirección IP, el nombre completo incluyendo el dominio y finalmente el nombre con el que la queremos conocer. De este modo, cuando conectemos con esa máquina, podremos usar *ftp nombre* en lugar de *ftp nombre_completo.*

Un poco más de atención merecen las dos últimas líneas. La primera indica la dirección que posee nuestro dispositivo de loopback. Este es un dispositivo, digamos virtual, que conecta nuestra máquina consigo misma. Esto, que al principio parece una tontería puede ser muy útil para comprobar el funcionamiento de algunos programas.

La siguiente línea indica la dirección que tendrá nuestra máquina. Como a priori no la sabemos, si pusiéramos un valor cualquiera nos encontraríamos con que no acertaríamos en la mayoría de las veces. Por ello colocamos 0.0.0.0, un número llamado mágico, porque indica al sistema que utilice la dirección que le provea el servidor PPP correspondiente.

Conclusiones finales

Como hemos podido ver a lo largo de este artículo, la conexión de nuestro ordenador a la red Internet bajo Linux no es especialmente complicada, y tan sólo tendremos que entender un poco cómo funciona la conexión.

De hecho, podremos automatizar todo lo anteriormente desarrollado de una manera más sosegada en un único script:

```
#!/bin/sh

DEVICE=módem

NUMERO=numero_del_proveedor

if [ -f $/var/spool/uucp/LCK..$DEVICE ]
then
      echo "El módem esta ocupado"
      exit 1
fi

pppd
```

Si no hubiéramos incluido las distintas opciones en el fichero */etc/ppp/options,* o si quisiéramos incluir otras adicionales, las colocaríamos tras *pppd* en el script anterior.

Referencias adicionales

- Infobia COMO
- Spanish Linux HOWTO
- PPP Howto
- Guía de Administración de Redes en Linux
- Linux Serial HOWTO
- NET 3 HOWTO

Programación de DirectX con Delphi 2.0 (I)

Constantino Sánchez Ballesteros

Comenzamos esta sección con el estudio y uso de las famosas librerías DirectX de Microsoft. Con la ayuda de un compilador, en este caso Delphi 2.0, el soporte que nos brinda Windows 95 y las propias DirectX, seremos capaces de crear vídeojuegos realmente profesionales.

Como es lógico para cualquier programador, el camino que queda por recorrer hasta completar el objetivo deseado no es todo lo agradable que uno quisiera, pero el esfuerzo empleado valdrá la pena ¡os lo aseguro!

Requisitos para empezar a trabajar

DirectX se utiliza con Windows 95, ya que Windows NT suele dar bastantes problemas en los accesos a memoria de vídeo debido a sus restricciones. Por supuesto, para programar se necesita un equipo bastante potente, y puesto que utilizaremos Delphi, lo mínimo recomendable sería:

- Pentium 166 Mhz
- 16 Mb de RAM
- Tarjeta gráfica SVGA
- Tarjeta de sonido de 16 bits compatible

Lógicamente, podemos intentarlo con un equipo menos potente, pero el tiempo que se pierde en las cargas de programas, las *DLLs* (librerías dinámicas de enlace), compilaciones, etc., puede llevarnos a la desesperación a más de uno.

Un poco de historia

Atrás quedaron los quebraderos de cabeza en la programación de vídeojuegos bajo el paleolítico MS-DOS. Los que programaban juegos bajo este sistema operativo (entre los que me incluyo), debían conocer su estructura interna al máximo, puesto que para el uso del teclado, música, etc. se necesitaba un amplio conocimiento de las interrupciones del DOS. Ya no quiero ni hablar de la segmentación de memoria,

Logotipo de aplicaciones que usan Drect.



puesto que para utilizar la memoria expandida o extendida, necesitábamos controladores especiales además de saber programarlos adecuadamente.

Windows 95 es la plataforma ideal para la programación de vídeojuegos

Cuando debíamos crear una rutina de *scroll* rápida, teníamos que pelearnos directamente con el lenguaje de bajo nivel por excelencia: el Ensamblador. ¡Ahí sí que venían los problemas uno detrás de otro! Si necesitabas efectuar transparencias en algún *sprite*, pensabas ¿y cómo se hace eso?

Todos estos problemas comentados, y muchos más, desaparecen por completo mediante el uso de las librerías DirectX de Microsoft, puesto que nos permiten utilizar su amplio abanico de funciones sin importarnos en absoluto la estructura interna del sistema operativo y sin necesidad de utilizar ensambladores para crear las rutinas críticas de nuestros programas. Si a este hecho le sumamos el poder utilizarlo con lenguajes de alto nivel, ¿qué más se puede pedir?

Qué componentes nos ofrece DirectX

Básicamente, tenemos a nuestra disposición cinco módulos bien diferenciados:

- DirectDraw
- Direct3D
- DirectPlay
- DirectInput
- DirectSound

Cada uno de ellos tiene sus peculiaridades y se basan en la realización de tareas muy diferentes. A continuación, descubriremos las funcionalidades de cada módulo, uno por uno. También se puede consultar el artículo que sobre el tema se publicó en Sólo Programadores nº 37.

DirectDraw

Permite la manipulación directa de la memoria de vídeo, *blitters* por hardware, *overlays*, *page-flipping*, rotaciones, etc. DirectDraw proporciona estas funcionalidades manteniendo compatibilidad con aplicaciones Windows 95 y sus drivers. DirectX se construyó con la intención de dar velocidad a las operaciones gráficas y, por lo tanto, DirectDraw no es un API de alto nivel. Lo interesante es que nosotros sí podemos trabajar en lenguajes de alto nivel al utilizarlo.

Este API nos proporciona acceso directo al hardware de las tarjetas gráficas, mientras mantiene total compatibilidad con el GDI de Windows. DirectDraw trabaja con una amplísima variedad de tarjetas gráficas (prácticamente todas las que van saliendo al mercado) y abarca desde simples SVGA hasta hardware avanzado (tarjetas aceleradoras 3D) que proporcionan *clipping*, *stretching* y soporte para paletas gráficas diferentes a las típicas RGB. La interfaz se diseñó para que las aplicaciones pudieran tomar todas las capacidades del hardware subyacente y utilizarlas de la forma oportuna.

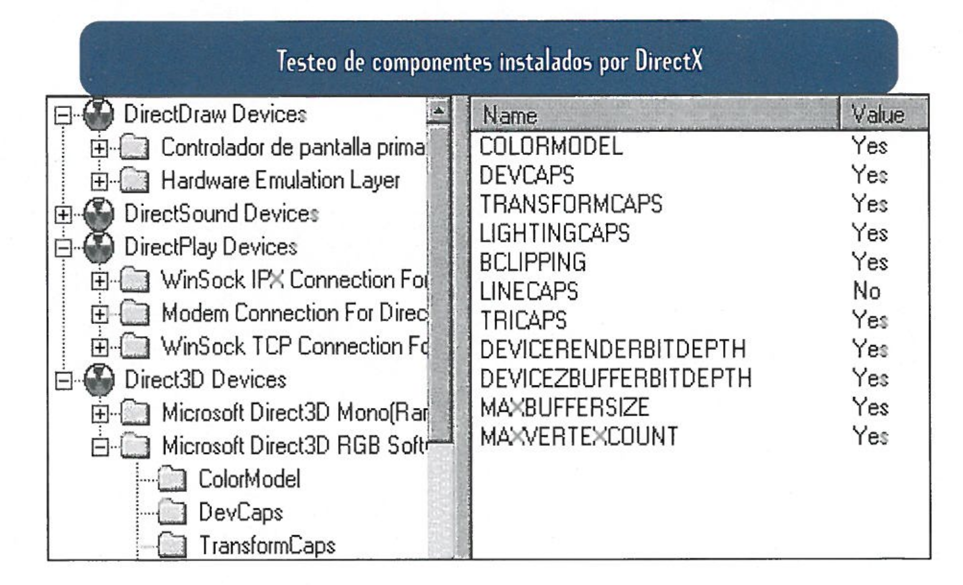
Estas son algunas de las cualidades que nos ofrece DirectDraw:

- Soporte de doble y triple *buffer*, además de *page-flipping*.
- Permite el acceso y control total de la tarjeta gráfica.
- Soporta *buffer Z* para las operaciones de 3D.
- Posibilita una alta calidad de gráficos proporcionando un acceso directo a las operaciones de *stretch* por hardware.
- Acceso simultáneo tanto a la memoria RAM, como la incluida en la propia tarjeta gráfica.

Mediante DirectDraw no necesitamos conocer los procedimientos necesarios para operaciones de *blit*, o modificación de registros de la paleta gráfica, ya que tan sólo es necesario llamar al procedimiento correspondiente.

DirectDraw HAL (capa de aceleración hardware)

DirectDraw HAL depende exclusivamente del hardware y contiene únicamente código específico para ese hardware. El HAL puede ser implementado en 16, 32 bits o una combinación de ambos mediante Windows 95. Sobre Windows NT siempre trabaja a 32 bits. Puede ser una parte integral del driver de la tarjeta gráfica o



Testeo de componentes instalados por DirectX

una *DLL* que comunica con el driver a través de una interfaz privada definida por el fabricante.

DirectDraw HAL es implementado por el fabricante de la tarjeta gráfica. Sólo implementa código dependiente del dispositivo instalado y no permite emulaciones de ningún tipo. Si una función no se puede crear por hardware, el HAL notificará que no existe capacidad hardware para la misma.

- **DirectDraw HEL (capa de emulación hardware)**

DirectDraw HEL presenta sus capacidades a DirectDraw justamente como debería hacerlo el HAL. Examinando dichas capacidades durante la inicialización de la aplicación, podemos ajustar los parámetros necesarios para obtener el máximo rendimiento sobre una diversidad de plataformas. Si DirectDraw HAL no está presente o una función específica no está diseñada por el hardware, DirectDraw emulará esa funcionalidad no soportada.

- **Tipos de objetos en DirectDraw**

Los objetos de DirectDraw representan el dispositivo gráfico. Puede haber un objeto DirectDraw por cada dispositivo gráfico lógico en la operación. Un entorno de desarrollo de aplicaciones puede utilizar dos monitores, uno de ellos ejecutando la aplicación utilizando DirectDraw y otro usando el GDI de Windows.

Un objeto DirectDrawSurface representa una región lineal de la memoria de vídeo que puede ser direccionada y manipulada. Estas direcciones de memoria de vídeo apuntan a *buffers* visibles de memoria (*primary surface,* superficie primaria) o *buffers* no visibles (*off-screen* o *superficies overlay*). Estos *buffers* no visibles suelen residir en la memoria de vídeo, pero pueden ser creados en la memoria del sistema si se necesita por temas de diseño hardware o emulación software de DirectDraw. Un *overlay* es una superficie que puede ser visible sin alterar los píxeles que oculta. Los *sprites* y *overlays* son sinónimos. Un mapa de texturas es una superficie que puede ser asignada sobre un objeto 3D.

DirectDrawPalette representa una paleta de 16, 256 o 65535 colores indexados. Las paletas proveen de colores a las texturas, superficies *off-screen* y *overlays*, sumando el hecho de que no se requiere tener la misma paleta en cada uno de ellos que la asignada a la superficie primaria.

Con DirectDraw se pueden crear objetos DirectDrawSurface, DirectDrawPalette, y DirectDrawClipper. DirectDrawPalette y DirectDrawClipper deben ser enlazados a los objetos DirectDrawSurface que afecten de forma directa. DirectDrawSurface puede requerir el enlace de varios DirectDrawPalette pero no es muy usual debido a que la mayoría del hardware actual no soporta paletas múltiples.

DirectDraw es el componente más importante de todo el sistema DirectX. Sin él, la visualización de cualquier escena sería imposible

Direct3D

Existe una estrecha relación entre DirectDraw y Direct3D, puesto que éste último necesita del primero para poder visualizar el trabajo efectuado. DirectDraw es el motor de Windows para la visualización de gráficos 2-D, 3-D, y vídeo. Si utilizáis Direct3D en la programación, deberéis familiarizaros también con DirectDraw.

El HAL se aprovecha de rasgos especiales como el *blitting* y *page flipping* (siempre a nivel hardware). Como hemos comentado anteriormente, el compañero de este sistema es el denominado HEL. Éste contiene procedimientos software de toda la funcionalidad disponible en DirectDraw y Direct3D. La combinación del HEL y el HAL permite la creación de aplicaciones para aprovecharse de cualquier tipo de hardware instalado sin sacrificar la independencia del dispositivo en cuestión. De este modo, los programadores pueden escribir código de una manera estándar con la satisfacción de que su programa correrá tan rápido como lo permita el sistema.

El módulo de renderizado de Direct3D se compone a su vez de otros tres módulos:

1. Iluminación
2. Transformación
3. Rasterización

Cada uno de éstos puede emularse mediante software, puede ser acelerado vía hardware y puede intercambiarse (en tiempo de ejecución si fuera necesario) para aplicaciones diferentes que requieran las mismas necesidades. Aunque parezca lo contrario, Direct3D contiene dos paquetes 3-D en uno. Los programadores pueden elegir el uso del API de alto nivel (Modo retenido), o el de bajo nivel (Modo Inmediato).

- **Retained Mode (Modo retenido)**

El Modo Retenido se diseñó para mantener servicios de alto nivel en la manipulación de escenas y objetos. Está colocado por encima del Modo Inmediato y tiene un constructor de geometrías, para que no necesitemos guardar bancos de datos de objetos y sus estructuras internas. Este *Engine* también apoya capacidades para la animación de *key-frames*. Con unas pocas llamadas a varios procedimientos, podemos cargar los objetos creados y los manipularlos dentro de las escenas con cierta facilidad. Esto conlleva diversos pros y contras:

- Pros: No necesitamos saber demasiado de la programación 3-D, además de ponernos a trabajar rápidamente y escribir menos código en la creación de efectos 3D.
- Contras: No es tan flexible como el Modo Inmediato.

Si sólo necesitamos crear funcionalidades 3-D en nuestra aplicación o no tenemos demasiado tiempo para desarrollar nuestras propias rutinas de geometría compleja, la elección será el modo retenido.

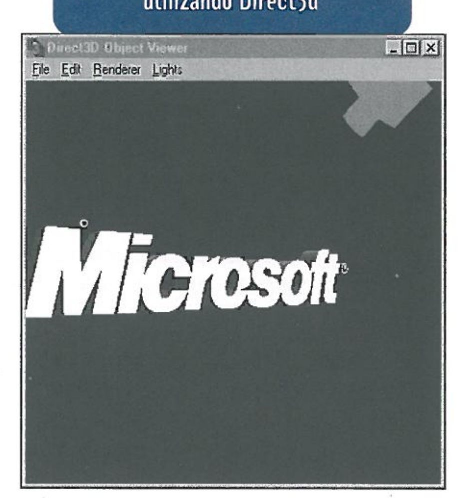
Inmediate Mode (Modo inmediato)

Proporciona una comunicación a bajo nivel del hardware acelerador. Éste es el API que utilizan los diseñadores que tienen una experiencia considerable en la programación 3-D.

- Pros: Conseguimos crear nuestras propias escenas en la dirección que necesitemos. Esto dará a nuestra aplicación más flexibilidad y mejor detalle si hemos perfeccionado las escenas para nuestras necesidades particulares.
- Contras: Debemos crear nuestra propia escena y necesitaremos tener muchos más conocimientos de programación en entornos 3D.

Para los programadores de videojuegos que necesiten un alto rendimiento en el entorno 3D (juegos como Quake, Blade, etc.) éste será el *Engine* a utilizar.

Renderización de un objeto utilizando Direct3d



Características de Direct3D

Este es un detalle a modo de resumen sobre las principales funciones que el API Direct3D puede hacer por nosotros:

- *Z-buffer*
- Uso eficaz de la memoria para la instalación de drivers
- *Flat* y *Gouraud* para el sombreado de modelos
- Puntos, líneas y modelos sólidos
- Múltiples fuentes de luz, direccionables y coloreadas
- Modelos de color Ramp y RGB
- Profundidades de color de 8, 16, 24, o 32 bits per pixel
- Vértices coloreados
- Múltiples puntos de vista, dispositivos, y cámaras
- Utilización de materiales
- Trazado de Texturas para cualquier tipo de sombreado
- Filtros Bilineal, trilineal y *mipmap* para la representación de texturas
- Corrección de perspectiva en las texturas para cualquier tipo de sombreado
- Transparencias, incluso en los mapas alfa de las texturas
- Deformaciones de objetos en tiempo real
- Fondos animados con información de profundidad optativa
- Efectos atmosféricos (como, por ejemplo, la niebla)
- Detección de colisiones, ...

A continuación, veremos algunos de los aspectos más destacables que ofrece el modo retenido, puesto que en definitiva, es el de uso más frecuente en el tema de videojuegos:

- Enlace de texturas, colores, y materiales a través de las jerarquías
- Sombras proyectadas
- Múltiples instancias de objetos y jerarquías
- *Keyframer* para la realización de animaciones en tiempo real

Si utilizamos el *keyframer*, podemos animar la posición y orientación de objetos, luces y demás componentes que representen la escena. Para animarlos, se crea un bucle que repite llamadas a unas funciones y el sistema calcula la posición que debe tener cada objeto interpolando los valores importantes. Pueden agruparse objetos de la animación y se pueden animar todos ellos de forma simultánea.

Lo que hay que saber...

Para poder comprender mejor el funcionamiento de Direct3D, debemos tener conocimiento de una serie de conceptos primordiales para que, a la hora de afrontar proyectos con este API, no nos suene nada a "chino":

- Dispositivos: en Direct3D, un dispositivo representa el despliegue visual para aplicar el render. Los dispositivos se pueden aplicar directamente a la pantalla, a las ventanas, o a la propia memoria del ordenador.

- Modos de color: el Modo retenido se apoya en dos modos de color para la representación de los objetos en las escenas 3D.

  1. RGB
  2. Modo monocromo (también llamado Ramp)

- Caras: una cara es un sólo polígono en una malla. Los métodos que contiene el API Direct3D nos permiten construir las caras de los vértices y asignarles el color, la textura, y los materiales que deseemos. Pueden guardarse caras en series, para realizar funcionamientos idénticos en caras múltiples.

- Marcos: se usan marcos para posicionar los objetos dentro de las escenas. Cada objeto tiene un marco, y a su vez, los marcos múltiples constituyen una escena. Los marcos se crean con una jerarquía, donde la propia escena es el rango más alto. Los métodos nos permiten crear marcos, agregar y anular sus jerarquías, además de conseguir información sobre las ramificaciones de ésta.

- Luces: Se asignan luces a los marcos para iluminar los objetos visuales dentro de la escena. Disponemos de cinco tipos de iluminación:

  1. Ambiente
  2. Direccional
  3. Paralela
  4. *Point*
  5. Reflectora (*Spotlight*)

- Materiales: los métodos nos permiten asignar las propiedades especulares de un material, con lo cual podemos definir la forma en que una superficie refleja la luz.

- Mallas: son colecciones de vértices y caras poligonales que constituyen un objeto. Los métodos para la manipulación de mallas incluyen aquellos para agregar vértices, caras y normales, así como los colores fijos, texturas, y materiales a una cara o a un grupo de ellas.

- Sombras: un objeto sombreado puede ser creado especificando la fuente de luz, el objeto que lanza la sombra, y el plano en el que la sombra se proyecta.

- Texturas: pueden mapearse texturas hacia las caras o pueden darse directamente como calcomanías. Las rutinas para la manipulación de texturas incluyen métodos para conseguir texturas coloreadas, control para asignar cómo se asignarán las texturas a las superficies, y grados de transparencias. Además, la calidad de texturas puede ser mejorada usando la técnica del *mip-mapping*, que proporciona las resoluciones de diferente calidad dependiendo de la distancia del objeto (de este modo se elimina la pixelación).

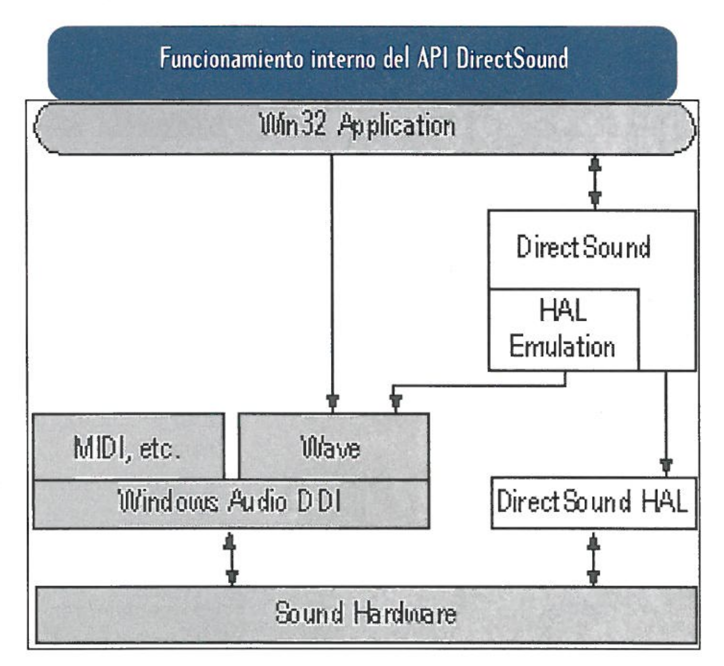
- Viewports: El viewport define la vista final de una escena después de que se han convertido las coordenadas 3-D a 2-D para el despliegue hacia la pantalla. Los métodos del control de cámaras asignan la posición, dirección, y orientación del viewport. En el interface del viewport también se especifican las transformaciones reales de 3-D a 2-D.

- Envolturas: las envolturas definen cómo una textura se mapea sobre un objeto visual. Tenemos a nuestra disposición varios modos de envolturas de texturas para los objetos:

  - Cilíndricas
  - Esféricas
  - Cromo
  - Flat

DirectSound

Este API es el componente de audio de DirectX que nos permite realizar *mixing* (mezclado simultáneo de sonidos y música), aceleración por hardware y acceso directo a la tarjeta de sonido. DirectSound obtiene esta funcionalidad mientras presenta una total compatibilidad con las aplicaciones basadas en Windows 95. Todos estos beneficios nos posibilitarán obtener el máximo rendimiento de las capacidades de audio que nos ofrezca cualquier tarjeta de sonido genérica compatible.



DirectSound puede emular mediante software las características de una tarjeta de sonido que no soporte directamente alguna funcionalidad especial.

Mixing

Esta es una de las capacidades más utilizadas de DirectSound. Nuestro programa puede crear uno o más *buffers* secundarios y escribir en ellos datos de audio. Podremos reproducir o parar cualquiera de estos *buffers*. DirectSound mezcla todos los *buffers* que se estén reproduciendo y traslada el resultado al primario, el cual pasa los datos a la tarjeta de sonido. No hay limitaciones en el número de *buffers* que pueden ser mezclados, exceptuando el tiempo que se consume en lo referente a procesamiento de datos por parte del microprocesador.

El Mixer permite al usuario que no experimente pausas al oír el sonido en el momento en que el *buffer* se reproduce y los altavoces lo emiten. En términos técnicos, significa que el sonido se transmite a los altavoces a menos de 20 milisegun-

dos. De este modo, si nuestra aplicación comienza a reproducir un *buffer* y a su vez una animación en pantalla, audio y vídeo aparentarán reproducirse de forma sincronizada.

Si utilizamos el HAL con DirectSound (esto es, cuando el driver de DirectSound para la utilización directa de hardware no esté presente), el Mixer no funcionará del modo anteriormente mencionado y la transmisión de sonido se efectuará en torno a los 100-150 milisegundos. Por ello, es aconsejable tener un buen hardware que acelere las operaciones sin necesidad de que intervenga el microprocesador.

Debido al hecho de que sola una aplicación puede reproducir mediante DirectSound en el momento de su ejecución, habrá conflictos si se intenta proceder con varios programas abiertos a la vez y utilicen el API.

Aceleración hardware

DirectSound automáticamente toma los recursos de la aceleración hardware referentes al sonido, incluyendo el Mixing y la memoria hardware que alberga los *buffers* de sonido. Nuestro programa no necesita saber del hardware instalado para utilizar la aceleración hardware. Este hecho se cumple también con los demás componentes de DirectX. De cualquier modo, podemos sacar el mayor provecho a los recursos hardware instalados en el ordenador recopilando una descripción de las capacidades hardware que tenemos mediante el uso de ciertas funciones. Con esta información, podemos especificar que *buffers* de sonido deberían recibir la aceleración hardware.

Dado que nosotros decidimos qué efecto de sonido utilizar, cuándo reproducir cierto *buffer* de sonido y qué prioridad tendrá cada uno, podemos localizar los recursos hardware existentes para las necesidades de cada uno.

Acceso directo al buffer primario

El *buffer* primario es el encargado de enviar los muestreos de audio a la tarjeta de sonido. DirectSound permite acceso directo al *buffer* primario. De todas formas, esta característica tiene un uso muy limitado a aplicaciones que requieren *mixing* especializado u otros efectos no soportados por *buffers* secundarios.

Los cortes en el tema de sonido son difíciles de corregir cuando una aplicación escribe directamente al primario y, aquellas que lo hacen, están sujetas a unos restringidos requisitos.

Un *buffer* primario es normalmente muy pequeño y si nuestra aplicación escribe directamente en él, debe hacerlo mediante bloques de datos a cortos intervalos para evitar que los bloques previos del *buffer* se repitan.

Durante la creación del mismo, no podemos especificar su tamaño y debemos aceptar la capacidad retornada por el API una vez se crea.

Cuando efectuamos acceso directo al primario, muchas capacidades de DirectSound se deshabilitan. Como muestra, baste decir que los *buffers* secundarios no son mezclados con el Mixer y, consecuentemente, la aceleración hardware del *mixing* desaparece (Cuando DirectSound mezcla sonidos de *buffers* secundarios, coloca los datos obtenidos en el primario).

La mayoría de nuestras aplicaciones deberían usar *buffers* secundarios para el acceso directo al primario.

Podemos escribir datos fácilmente en el secundario porque al utilizar grandes tamaños de bloques, obtenemos más tiempo para enviar el siguiente bloque de datos, minimizando el riesgo de cortes en el audio. Si nuestra aplicación necesita simples requerimientos de audio, obtendremos mejor rendimiento utilizando *buffers* secundarios para reproducir sonido.

DirectPlay

Este API simplifica el acceso de nuestro programa a los distintos servicios de comunicación. Primeramente fue desarrollado para la comunicación de juegos, aunque se podría utilizar para otro tipo de aplicaciones que necesitaran comunicaciones independientemente del protocolo utilizado.

Claro está que los juegos son mucho más divertidos si son jugados contra jugadores reales, y debido a este hecho, el PC ha experimentado más avance en las opciones de conectividad que cualquier otra plataforma en la historia.

DirectPlay nos evita conocer la complejidad de los diversos tipos de conectividad, para así centrarnos solamente en el desarrollo de videojuegos.

● Arquitectura de DirectPlay

DirectPlay utiliza para la implementación de conectividad un simple modelo de comunicación "enviar/recibir". La arquitectura DirectPlay requiere 2 tipos de componentes para funcionar:

1. El propio DirectPlay
2. Proveedor de servicios

DirectPlay lo provee Microsoft y presenta un interface común para el videojuego. El proveedor de servicios nos da soporte de comunicación para la utilización de DirectPlay en Red.

Cuando diseñamos un juego que utiliza el API, sólo necesitamos saber unas concretas funciones del mismo y utilizar las vías de comunicación que vayamos a implementar (modem, Red local o servicio online).

DirectPlay utilizará cualquiera de estos servicios siempre que estén instalados en el sistema del usuario. El juego interactúa con DirectPlay. Éste, a su vez, interactúa con uno de los proveedores de servicios instalados, y éste último utiliza el protocolo apropiado para establecer la comunicación.

Funcionamiento interno del API DirectSound



DirectInput

xDirectInput nos permite un rápido y consistente acceso a los joysticks analógicos y digitales, además del ratón y el teclado. Este API utiliza el registro para guardar valores de joysticks estándar, información de la calibración previamente configurada y valores de joysticks tipo OEM. La nueva versión 5.0 de DirectInput permite soporte para el novedoso Joystick *force feedback*, que consiste en enviar unos impulsos a la palanca del mismo para sentir en nuestra mano los desenlaces producidos en el juego. Por ejemplo, si en un juego de coches tomamos una curva, notaremos quecuesta mover la palanca debido a la fuerza centrífuga que nos proporciona ésta, o si nos dispara algún objeto, notaremos el impacto al instante.

DirectInput también permite el acceso a otro tipo de dispositivos más complejos, como pueden ser:

- *Touch screens* (soporte muy utilizado en los ordenadores portátiles)
- Tabletas digitalizadoras
- *Light pens* (lápices ópticos)

Las capacidades extendidas que nos permite este API pueden llegar al control de pedales, *flight yokes*, cascos de realidad virtual y otros dispositivos avanzados. Cada dispositivo puede utilizar seis ejes de movimiento diferentes, un punto de vista y 32 botones. Como podéis apreciar, el soporte que nos brinda DirectInput es realmente extenso.

Para determinar las capacidades de los dispositivos instalados en el sistema, tenemos a nuestra disposición varias funciones que retornan automáticamente unos valores que indican las posibilidades que tiene cada uno.

Delphi Games Creator (DGC)

Microsoft ha diseñado el API de DirectX de modo que puede utilizarse con cualquier compilador estándar, de los que existen en la actualidad, de C/C++ o Visual C++ para Windows.

Con estas premisas, en principio parece que es imposible utilizar el API con nuestro compilador favorito, Delphi 2.0. Como todo tiene solución, ciertas personas han diseñado la estructura creada en C para su utilización con Delphi. De este modo, estamos en disposición de utilizar todas las posibilidades de DirectX sin ninguna limitación.

Como complemento a esto, también se han creado una serie de componentes en Delphi, que utilizan DirectX de una manera más visual y fácil de programar. Estos componentes forman el llamado Delphi Games Creator (DGC) y disponen de utilidades para el manejo de bancos de imágenes, *sprites*, demos, editores de sonido, etc.

Delphi Games Creator nos proporciona el soporte necesario para la utilización de DirectX bajo Delphi 2.0

El colofón a este hecho es que los componentes son totalmente gratuitos y de libre distribución (Freeware), y por lo tanto, su utilización no implicará ningún tipo de obligación por nuestra parte. Los autores que han escrito estos componentes son John Pullen, Paul Bearne y Jeff Kurtz.

Instalación de Delphi Games Creator

Para proceder a la instalación de los componentes en el entorno de Delphi, debemos comprobar primero los siguientes detalles:

- Tener desinstaladas versiones previas del DGC (si este fuera nuestro caso).
- No tener ninguna otra versión de los ficheros principales que conforman la estructura de DirectX en nuestro *Path* de librerías (ddraw.pas, dsound.pas etc.).
- Tener instalado DirectX 3 o superior en nuestro PC.
- Necesitamos efectuar una copia de seguridad del fichero "*cmplib32.dcl*" localizado en el directorio BIN de Delphi. Esto se hace, por si hubiera problemas en la instalación de los componentes, para poder volver a restaurar Delphi copiando otra vez el fichero "*cmplib32.dcl*" al directorio BIN.

Habiendo efectuado las comprobaciones anteriores, procedemos a la instalación propiamente dicha:

1. Descomprimir el archivo zip *"dgc.zip"* al directorio que creamos conveniente.
2. Seguidamente ejecutaremos el fichero *"setup.exe"* para que comience la descarga de archivos en el directorio elegido.
3. Seleccionamos la opción *"Install"* de la barra *"Component"* del menú principal de Delphi.
4. Agregamos el fichero *"dgcreg.pas"*, el cual corresponde al fichero de registro de los nuevos componentes.
5. Pulsamos con el ratón sobre *OK* para construir la librería de componentes.
6. Si los componentes han sido instalados correctamente, una pestaña con la inscripción DGC aparecerá en la paleta de componentes.
7. Debemos asegurarnos de que la opción llamada *'Break on Exception'* esté desactivada. Esta opción se encuentra en el menú de Delphi *Tools/Options*.

Si durante la instalación de los componentes aparece el error **"1157"**, significará que no están instalados los drivers de DirectX 3.

Detalle de los componentes instalados con DGC

A continuación veremos las características principales y la utilidad de los componentes instalados con DGC:

TDGCAUDIO
Se encargará de inicializar automáticamente DirectSound y cargará en memoria la librería de sonido adjuntada por el programador. Este es probablemente el componente de DGC más fácil de utilizar, y puede ser usado con una ventana normal de Delphi para cualquier tipo de aplicación.

TDGCHISCORE
Este componente nos brinda un camino fácil para la creación de tablas de récords en nuestro juego.

TDGCIMAGELIB
Básicamente es utilizado para guardar un grupo de imágenes conjuntamente con nuestro archivo de programa ejecutable. DGCImageLib se utiliza asignándola en la propiedad correspondiente del componente TDGCScreen.

Las imágenes son cargadas dentro del componente utilizando el editor de propiedades. La librería de imágenes deberá ser creada mediante el editor que tendremos instalado con los componentes.

El uso de componentes en Delphi permite una programación visual y modular con un par de pulsaciones de ratón. Con los componentes DGC la programación de DirectX será un puro trámite.

TDGCINTROLIB
Este componente se utiliza para poder generar transiciones y otros efectos especiales similares con ayuda del editor FX (de efectos).

TDGCJOYSTICK
Permite un fácil acceso de cualquier programa a los joysticks que se encuentren conectados, ya sean estos de tipo digital o analógico.

TDGCPLAY
Encapsula el objeto DirectX DirectPlay y permite el acceso de los diversos juegos a Internet, redes locales y módem.

TDGCSCREEN
Es el componente principal del Delphi Games Creator. Inicializa automáticamente DirectDraw y crea un sistema de páginas con doble *buffer* a la resolución de pantalla especificada por nosotros.

Para utilizar el componente, debemos colocarlo en nuestro formulario de proyecto, elegir la resolución y tendremos control total de las superficies creadas con DirectDraw para poder utilizar nuestras propias rutinas gráficas, las creadas con la librería de imágenes o la clase *Tcanvas* ¡Todo un lujo!

TDGCSOUNDLIB
Básicamente se utiliza para guardar sonidos con nuestro programa ejecutable. Los sonidos son cargados dentro del componente utilizando el editor de propiedades. La librería de sonido se debe crear mediante el editor de librerías de sonido incorporado por DGC.

TDGCSPRITEMGR
El *"Sprite manager"* y la *"Class sprite"* hacen posible el crear animaciones y movimientos .

TDGCSTARFIELD
Permite la creación de campos de estrellas con *scrolling* en 2D. Este campo de estrellas puede ser coloreado, con efecto especial de *flash*, y movible en cualquier dirección.

Primer contacto

Esta primera entrega nos ha servido para ir tomando contacto con el material con el que vamos a trabajar. En el próximo fascículo crearemos nuestro primer programa con DirectX bajo Delphi e iremos viendo el uso de las diferentes utilidades adjuntadas por el DGC (editor de imágenes, sonidos, etc.)

WWW

Visual JavaScript: El Paso Necesario

Alejandro M. Reyero Abad
axl@las.es

Todo programador que desee implementar funciones avanzadas en sus páginas HTML sabe que JavaScript es la opción necesaria: Es sencillo, potente y rápido (si lo comparamos con un Applet Java normal, por ejemplo).

El problema de JavaScript (hasta ahora) era la necesidad de programar a la antigua usanza, esto es, con un editor de texto estándar, ejecutando la aplicación a continuación en el browser de turno, verificando errores, editando de nuevo "a pelo", ejecutando, etc. La ausencia de un autentico entorno de programación (sea o no visual) frenaba la expansión de JavaScript en ámbitos profesionales.

"¿JavaScript en ámbitos profesionales? ¿Para qué?, si eso solo sirve para hacer tus WEBs más bonitos..." Se preguntará mas de uno. Veamos posibles escenarios de implantación de aplicaciones JavaScript.

El más evidente es la página WEB, personal o corporativa, que incluye pequeños scripts en JavaScript para ofrecer un cierto grado de interactividad con el usuario, presentándole un *alert* cuando deja el WEB o haciendo que una imagen cambie cuando el usuario pasa con el cursor del ratón por encima.

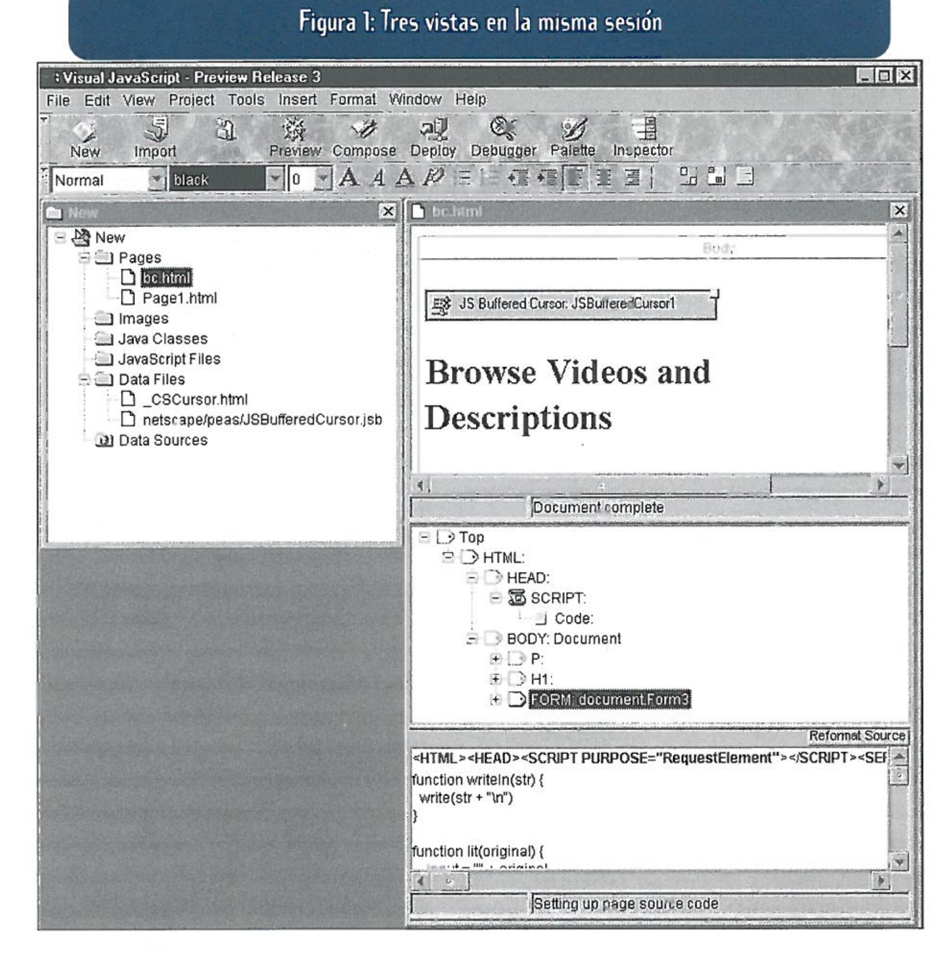
Pero imaginemos algo mas profesional. Supongamos que una empresa importante, con delegaciones en toda España y Suramérica, quiere pasar de su sistema de control de datos descentralizado a un sistema de datos centralizado, sistema al que las delegaciones acceden mediante una Intranet para tomar y ofrecer datos. La manera más evidente de hacer esto es mediante la WEB, ya que no importa el sistema operativo (Windows, Mac, Unix...), máquina o plataforma de los ordenadores que accedan al sistema, todos tienen su browser con soporte JavaScript (Netscape Navigator). Esto supone un importantísimo ahorro en infraestructuras, al utilizar la base informática ya instalada.

Tomemos como ejemplo el caso de una empresa de transportes. En cada ciudad, esta empresa tendrá, al menos, una delegación. Cada delegación recoge, envía y recibe paquetes de todo Hispanoamérica. ¿Cómo se controla tal ingente cantidad de datos? ¿Cómo sabe la delegación de México DF que el paquete que le llega de la delegación de Gijón ha llegado en condiciones (todos los bultos), que es a portes debidos y que el cliente que lo envía tiene crédito (se le cobran todos los envíos a fin de mes)? Con un sistema centralizado solo tendría que ir con su ordenador a la página WEB correspondiente, preguntar los datos de la expedición salida de Gijón el día X, comprobar que ha llegado todo en condiciones, e indicarle al sistema que todo está correcto.

Una aplicación de esta complejidad utilizaría JavaScript para comprobar que los campos introducidos en un FORM son correctos (que todo sean números en un

La versión 1.2 del lenguaje JavaScript ha supuesto un salto cuantitativo y cualitativo en el desarrollo de aplicaciones orientadas a Internet/Intranet. Netscape, sabedora de poseer un caballo ganador, intenta ahora con Visual JavaScript poner una herramienta de programación visual a disposición de los sufridos programadores.

Figura 1: Tres vistas en la misma sesión



campo numérico, que los campos NOT NULL no estén vacíos, que una fecha esté correctamente introducida...), le daría formato a las entradas tomadas de una base de datos, introduciría mejoras visuales, etc.

Estos mismos beneficios los obtendría una empresa mas modesta que cambiara su red Novell por una Intranet bajo TCP/IP (dada la facilidad para publicar documentos HTML) siendo Visual JavaScript la herramienta utilizada para el desarrollo de páginas con JavaScript.

Presentación de Credenciales

Visual JavaScript está dirigido a programadores empresariales que quieren desarrollar aplicaciones Intranet/Extranet estándar y multi-plataforma, utilizando componentes reutilizables con el mínimo esfuerzo. Estas aplicaciones estarán basadas en HTML, JavaScript y Java.

En principio, al ser una herramienta visual, no es necesario conocer estos lenguajes para desarrollar aplicaciones con Visual JavaScript. No obstante, cualquier programador que utilice entornos visuales sabe que esta aseveración está bastante alejada de la realidad.

Requerimientos del Sistema

Para desarrollar y ejecutar aplicaciones Visual JavaScript que utilicen JavaScript en el cliente y en el servidor (client-side y server-side JavaScript), necesitaremos una serie de herramientas, consistiendo el entorno de trabajo habitual en:

- Una plataforma de desarrollo, donde instalaremos Netscape Visual JavaScript y Netscape Communicator (4.01 o superior). Utilizaremos esta máquina para crear aplicaciones Visual JavaScript (la ultima versión, la PR4, está disponible para Windows 95/NT y Solaris).

- Un servidor de desarrollo, donde instalaremos un servidor Netscape (para client-side JavaScript). Este servidor puede ser la misma máquina donde desarrollamos (o ser innecesario).

Si además nuestra aplicación utiliza el servicio de bases de datos de Netscape, LiveWire, necesitaremos:

- Un servidor de bases de datos relacionales en nuestro servidor de bases de datos. Normalmente se instala el servidor WEB y el de bases de datos en la misma máquina.

- El cliente de la base de datos y el software de red en el servidor WEB.

" El programador individual utilizará, en la mayoría de los casos, JavaScript únicamente en el cliente."

El programador individual utilizará, en la mayoría de los casos, JavaScript únicamente en el cliente, con lo que no tendrá que preocuparse de servidores ni de bases de datos.

El Modelo de Componentes

Los componentes de Visual JavaScript están basados en el modelo de componentes JavaBeans, desarrollado por Sun Microsystems para su lenguaje Java. Gracias a los JavaBeans, es posible desarrollar aplicaciones basadas en "servicios", que serán portables, podrán correr en cualquier plataforma, e interoperar entre ellos.

Los JavaBeans se definen por:

- Introspección: Permite que los beans expongan sus propiedades, métodos y eventos a Visual JavaScript.
- Personalización: Permite que la apariencia y comportamiento de un bean sean modificados.
- Persistencia: Permite que las propiedades y comportamiento de un bean aún permanezcan después de la personalización del mismo.
- Propiedades: Permite la personalización de características como el color, el tamaño o la fuente.
- Eventos: Permiten que los beans se conecten entre ellos.

Nuestro Primer Programa en Visual JavaScript

Para ejecutar Visual JavaScript tendremos que tener primero instalado el Netscape Communicator (4.01 o superior) en nuestro ordenador.

Una vez en funcionamiento, vemos que Visual JavaScript organiza el código fuente en proyectos con extensión .prj, que no contienen los códigos fuente del proyecto en sí, sino que, por el contrario, apuntan a su localización.

Para crear un nuevo proyecto pulsaremos en la opción "New" en el menú "Project".

Tras escoger su nombre, el subdirectorio donde se va a guardar, y el subdirectorio donde se crearán los archivos temporales necesarios para los tests del proyecto, aparecerá nuestro el mismo en la ventana "Project".

En estos momentos nuestro proyecto está vacío. Podremos añadir páginas ya creadas importándolas, utilizando el botón "Import" de la barra de herramientas. También se puede añadir una página en blanco, pulsando "New" en la barra de herramientas.

"Se pueden editar las páginas utilizando el programa Composer de Netscape Communicator."

Para editar una página únicamente tendremos que hacer un doble click en alguna de las páginas que ya habremos importado (o creado en blanco). Se puede editar y visualizar páginas de tres maneras diferentes:

- Layout: Esta vista muestra la página en modo "Lo que ves es prácticamente lo que tienes".
- Structural: Esta vista muestra la página de forma jerárquica. Se pueden arrastrar y soltar tags HTML dentro de esta vista.
- Source: Muestra la página entera en texto HTML puro y duro, para poder introducir Tags a mano.

Vemos en la Figura 1 las tres diferentes vistas en una misma sesión de Visual JavaScript.

Además del editor de páginas de Visual JavaScript, podremos utilizar Netscape Composer para editar las páginas pulsando el botón "Compose" en la barra de herramientas.

Añadiendo Componentes a la Página

Los componentes son los elementos que utilizamos para construir aplicaciones Visual JavaScript. La paleta es una colección de componentes que pueden ser utilizados para construir páginas HTML, y contiene elementos de formulario, bloques de texto, imágenes, y applets. Si pasamos el ratón por encima de un elemento de la paleta veremos una pequeña ayuda que describe el elemento; si hacemos un doble click llamaremos al "Inspector", que nos muestra las propiedades del componente.

Podremos arrastrar y soltar componentes de la paleta hacia nuestras páginas y viceversa. Si arrastramos un objeto del editor de páginas a la paleta lo añadiremos a la misma, permitiéndonos utilizarlo en otras páginas y aplicaciones. Por defecto, la paleta contiene una serie de elementos de formularios HTML, componentes basados en JavaScript, componentes LiveWire de acceso a bases de datos, y componentes Java.

Una vez finalizada la edición de nuestras páginas, y añadidos los componentes necesarios, podemos pulsar el botón de "Preview" para visualizarla en el Netscape Navigator. El comando "Preview" carga la página sin procesos basados en el servidor. Si nuestra página utiliza componentes "server-side", se visualizará de manera diferente a cuando la página esté compilada y situada en el servidor.

Utilizando el Inspector

Visual JavaScript nos permite modificar propiedades. Podemos soltar componentes en una página en la ventana de edición y modificar su comportamiento utilizando el Inspector.

Para inspeccionar las propiedades de un componente tendremos que hacer doble click en la paleta o en el editor de páginas. Si el inspector está ya abierto, simplemente tendremos que pulsar en un objeto en el editor o en el árbol del proyecto para ver sus propiedades.

Para cambiar las propiedades de un componente sólo tendremos que seleccionar el texto de la propiedad en la columna derecha del inspector.

Conectando Componentes

Cada vez que un botón es pulsado, o una caja seleccionada, se genera un evento. Para que estos eventos se relacionen con acciones concretas en nuestro programa, necesitaremos incluir un método que espere y responda al evento.

Una de las funcionalidades mas importantes de Visual JavaScript es la capacidad de especificar interacciones de alto nivel entre componentes y eventos. Las dos partes de toda interacción son el evento disparador y la acción. El evento disparador es el origen de la acción; la interacción entre ambos determina cuando ejecutar el código. Podremos añadir condiciones a una interacción introduciendo comandos "if" en nuestro código.

El creador de conexiones (Connection Builder) nos permite construir gráficamente relaciones entre componentes: Visual JavaScript genera automáticamente código para una relación especificada. Con Visual JavaScript, utilizamos el creador de conexiones para desarrollar la interacción entre el componente visual y los eventos correspondientes.

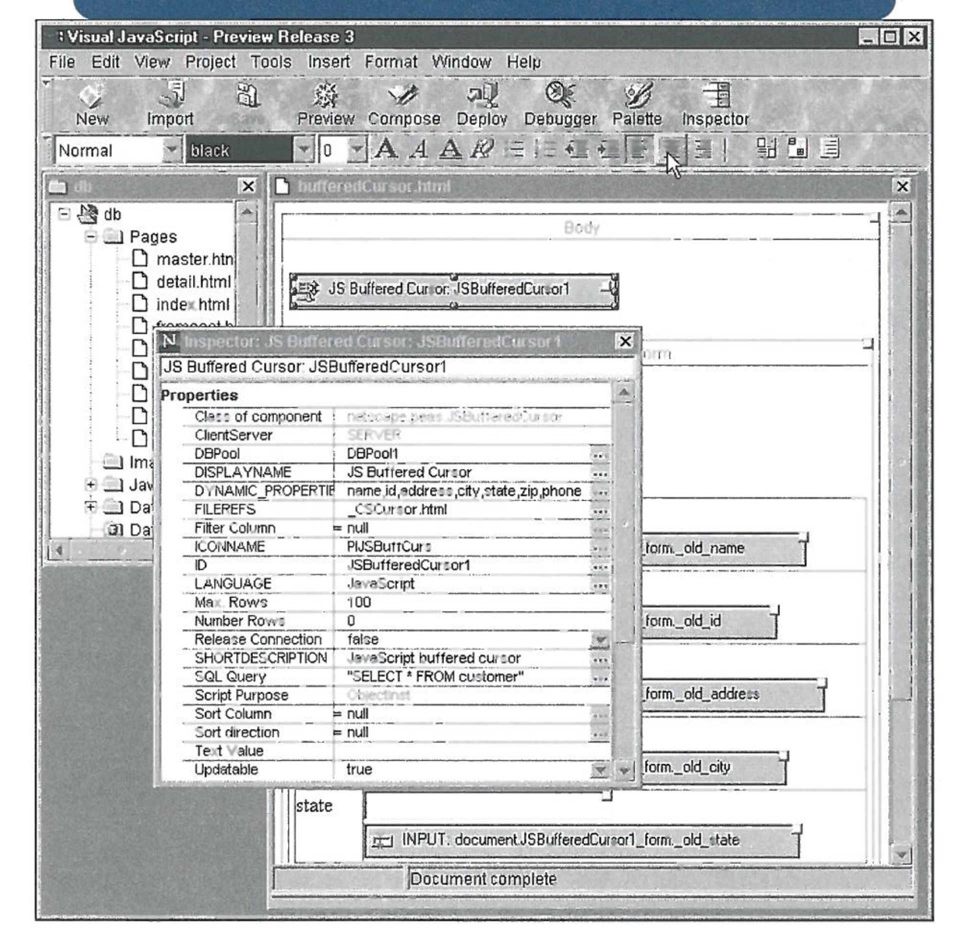


Figura 2: Inspección del objeto JSBufferedCursor"

Conclusiones

Visual JavaScript es el paso necesario para que la programación en este lenguaje se popularice finalmente en entornos profesionales, permitiendo el desarrollo de aplicaciones intranet de importancia. Hasta ahora solo hemos podido analizar versiones Beta del mismo, pero promete.

Bibliografía

- "El Modelo de Objetos de JavaScript", por Alejandro M. Reyero. Solo Programadores nº 32.
- "JavaScript: Frames y Windows", por Alejandro M. Reyero. Solo Programadores nº 33.
- "HTML Dinámico con Netscape Communicator y JavaScript (Hojas de Estilo)", por Alejandro M. Reyero. Solo Programadores nº 35.
- "HTML Dinámico con Netscape Communicator y JavaScript II (Layers)", por Alejandro M. Reyero. Solo Programadores nº 36.

Contactar con el autor

Si el lector quiere hacer llegar sus opiniones, preguntas, ejemplos, o, simplemente, decir "hola" al autor:

E-Mail: axl@las.es

WEB: http://xvt.allgames.com

Sistemas Distribuidos

Jorge F. Delgado Mendoza

Reproducir una película de vídeo, mandar un *fax*, escuchar nuestro CD de música favorito, todo ello mientras dictamos una carta a nuestro editor de texto. Aunque parezca increíble, los ordenadores personales de hoy en día son capaces de realizar tareas que, hasta hace muy poco, estaban reservadas a las sofisticadas estaciones de trabajo de *Silicon Graphics* que solo podíamos ver en las películas.

Por si esto fuera poco, la popularización de las redes de área local - especialmente la siempre presente *ethernet* - en todos los ambientes, han dado lugar a la aparición de aplicaciones que podría muy bien haber salido directamente de una novela de *Isaac Asimov* o *Robert A. Heinlein*.

Hoy en día es posible acceder a los archivos de un compañero que está en la otra punta del edificio, montar localmente sistemas de ficheros que pertenecen a otra máquina, hacer que nuestro ordenador reproduzca un vídeo en la pantalla de nuestro jefe y un millón de cosas más.

Cada día se globalizan más las comunicaciones. Poco a poco todas estas redes de área local se van interconectando a través de enlaces más sofisticados capaces de enviar grandes cantidades de información de un lado para otro. Lo que en su día fueron enlaces *RDSI* de 192 Kb/s se han convertido hoy en sistemas *Frame Relay*, *ATM*, etc, capaces de manipular transferencias de información del orden de los *Gigabytes* en el breve lapso de un segundo.

Sin embargo no es suficiente. Gradualmente vamos exigiendo más y más a los sistemas informáticos que tenemos encima de nuestras mesas. A pesar de que los microprocesadores multiplican su velocidad por dos en el breve espacio de un par de años y que la diferencia entre los sistemas compatibles PC y las estaciones de trabajo va siendo cada día más conceptual que real, siempre existe algún servicio o aplicación que no podemos utilizar porque nuestro ordenador no da más de sí.

Tenemos un claro ejemplo en los juegos para PC. El clásico *DOOM* de *Id Software* fue una revolución en su tiempo. Utilizando una resolución de 320x200 era capaz de presentar texturas en tiempo real en nuestra pantalla a casi 25 imágenes por segundo en un 486-33 con una tarjeta de vídeo *Vesa Local Bus*.

Hoy en día, ni siquiera el ordenador personal más potente es capaz de ofrecer, sin *hardware* especial, 25 imágenes por segundo en el *Quake* si utilizamos la resolución de 1024x768. Ya no es suficiente presentar texturas, es necesario realizar *bilinear filtering* para evitar que la imagen muestre cuadrados si nos acercamos mucho a una pared. Ya no basta con presentar una textura plana con ilusión de movimiento; es imprescindible realizar un modelo poligonal del objeto y mapear las texturas en sus caras.

Por otro lado, lo que en un principio eran aplicaciones aisladas, que como

Cada día se hacen más imprescindibles las redes, y exigimos más y más a nuestros sistemas. A medida que se produce este avance, las arquitecturas clásicas de comunicaciones van demostrando su ineficacia para manejar los crecientes recursos de los que disponemos. Los Sistemas Distribuidos son una solución a estos problemas.

mucho podía acceder a ficheros presentes en otros ordenadores montando localmente sistemas de ficheros remotos, se convirtieron en programas capaces de interaccionar con sus gemelos que se estuvieran ejecutando en otras máquinas de la misma red local.

Más tarde llegó el concepto de *Tiempo Real*. En el mismo *DOOM*, el nivel de diversión aumentaba radicalmente en las partidas multijugador, ya fuera a través del puerto serie o conectados a la red local. A fecha de hoy, raro es el juego que no permite que varios jugadores sean capaces de interaccionar en tiempo real a través de una red local.

Las exigencias en cuanto a conectividad están a la altura, e incluso más allá, de lo que le pedimos a nuestro ordenador. Al principio bastaba con tener la posibilidad de que dos, tres o cuatro usuarios utilizaran la misma aplicación. Hoy en día no sólo queremos poder simultanear un número ilimitado de usuarios, sino que además pretendemos salir más allá de los límites de nuestra propia red y ser capaces de interaccionar con personas que estén en el otro lado del mundo, todo ello gracias al boom de la *Internet*.

En resumidas cuentas, podríamos decir que cada día queremos más en tres áreas muy claramente diferenciadas:

- **Capacidad de Proceso**: nunca tenemos un ordenador lo suficientemente rápido, ni una tarjeta de vídeo suficientemente capaz. Queremos más ancho de banda en la red, menos latencia, más espacio en el disco, etc...
- **Conectividad**: ya no nos basta con comunicarnos con cuatro usuarios cuyas máquinas estén en el mismo segmento de red. Queremos hacer desaparecer la limitación espacial y encima queremos que el número de conexiones sea arbitrario.
- **Interfaz de Usuario**: hace mucho tiempo que despreciamos los interfaces de línea de comando. Hoy en día no es suficiente con un sistema de navegación capaz de presentar imágenes estáticas, queremos vídeo y audio, y lo queremos en tiempo real. Para qué utilizar imágenes de 320x200 en 8 colores si podemos obtener 640x400 píxeles en 16-bits.

Además, cada día se hace más necesario el poder conectarnos con máquinas cuya arquitectura sea diferente de la nuestra. Queremos poder trabajar con nuestro PC con *Linux* conectándonos a una base de datos *Oracle* que está situada en un servidor con *WindowsNT* mientras manipulamos la información allí obtenida con el programa *MacIntosh* de proceso de imágenes. Es decir queremos poder movernos en **Entornos Heterogéneos** de lo más variado.

Tradicionalmente este tipo de problemas se han resuelto de dos maneras opuestas, en las que la influencia de su entorno de origen queda muy patente. Estas dos arquitecturas son:

- *Sistemas Cliente/Servidor.*
- *Sistemas Diseñados* Ad-Hoc.

Sistemas Cliente/Servidor

Es un tipo de arquitectura originada en el mundo *Unix*, donde la conectividad era primordial y el interfaz de usuario poco importante, ya que prácticamente todos los usuarios estaban versados en el uso de sistemas informáticos. Algunos ejemplos clásicos de este tipo de aplicaciones incluyen:

- FTP: *File Transfer Protocol*. Un clásico de la *Internet* y una de las aplicaciones más utilizadas hasta la llegada de los navegadores, permite acceder a sistemas de ficheros remotos para poder leer o escribir información en ellos.
- Archie: Sistema de búsqueda de información en ordenadores remotos.
- Usenet News: Conjunto de aplicaciones destinado a distribuir y presentar la información de los foros de discusión de *Internet*.

Originado, como ya hemos dicho, en el mundo de las estaciones de trabajo *Unix*, este tipo de aplicaciones pre-

Figura 1.- Comparativa de los diferentes parámetros relevantes en los sistemas informáticos entre la Arquitectura Cliente/Servidor y los Sistemas Diseñados Ad-Hoc.

	Cliente/Servidor	*Ad-Hoc*
Capacidad de Proceso	*Media*	*Media*
Conectividad	*Alta*	*Baja*
Interfaz de Usuario	*Pobre*	*Cuidada*
Heterogeneidad	*Alta*	*Nula*
Transparencia	*Baja*	*Nula*

sentan un alto índice de conectividad, siendo capaces de admitir el acceso simultáneo de varios cientos de usuarios. Al emplear protocolos de comunicaciones estandarizados como son TCP/IP y UDP/IP, este tipo de sistemas es capaz de comunicarse con todo tipo de máquinas.

Sin embargo, no todo son ventajas. Este tipo de sistemas presenta los siguientes inconvenientes:

- El *Interfaz de Usuario* ha sido, tradicionalmente, muy pobre. Aunque poco a poco se van haciendo grandes progresos, las necesidades en cuanto a heterogeneidad y conectividad han primado siempre sobre la apariencia de las aplicaciones. Sin embargo, gracias a esfuerzos como *Motif*, *Java*, *VRML*, *HTML*, etc., las carencias en este aspecto son cada día menores.

- Más importante es otro defecto, inherente a la arquitectura *Cliente/Servidor*. Los sistemas diseñados a partir de las aplicaciones en red clásicas, están orientados hacia el establecimiento de enlaces *punto a punto*, lo que ocasiona una gran sobrecarga en cualquier tipo de red de difusión de datos, ya que es muy frecuente que la misma información circule por la red un número elevado de veces.

Sistemas diseñados Ad-Hoc

Su origen es el mundo del PC, donde presentar la información de una manera intuitiva, utilizando al máximo los pobres recursos del ordenador, era el objetivo principal. Las aplicaciones de este estilo son una evolución de los sistemas *Multimedia* monousuario del entorno de los ordenadores personales.

Introducidos por una u otra compañía de *software*, los sistemas son, generalmente, cerrados y propietarios, dificultando o incluso imposibilitando cualquier comunicación con otras aplicaciones que no sean aquellas para las cuales se diseñaron.

Estos sistemas, aunque muy vistosos y eficientes desde el punto de vista de los recursos del PC, presentan dos deficiencias muy importantes:

- Su *Conectividad* es muy pobre. Generalmente permite la conexión de un determinado número fijo de usuarios. El sistema de difusión que emplean para mantener la coherencia entre los datos de los distintos miembros de la red suele ser poco escalable, saturando la red si se intenta aumentar el número de usuarios.

- Al ser sistemas propietarios para PC's, su *heterogeneidad* es nula. Excepto en algunos casos excepcionales, es imposible incorporar al sistema usuarios que utilicen máquinas de arquitecturas diferentes.

Una posible solución: El Sistema Distribuido

Analizando la Figura 1, podemos llegar rápidamente a la conclusión de que ninguna de las dos soluciones está cerca de ser óptima. Para paliar las carencias de ambas aproximaciones vamos a proponer lo que se conoce actualmente como *Sistema Distribuido*.

Este tipo de sistemas se encuentra aun en desarrollo, aunque existen varios modelos operativos hoy en día. Como ejemplos de Sistemas Distribuidos podemos citar el Sistema Operativo *Amoeba*, y los Sistemas *Orca*, *PVM* y *Linda*.

Las ventajas de una Sistema Distribuido son numerosas, entre ellas - y ciñéndonos a las deficiencias observadas en los sistemas actuales - podemos destacar:

- **Conectividad**: un Sistema Distribuido puede utilizar cualquier tipo de protocolo de comunicaciones para comunicar a sus diferentes miembros entre sí. Gracias a esta peculiaridad podremos usar protocolos estándar como puede ser TCP/IP o UDP/IP, lo cual nos garantiza al menos la misma conectividad de los Sistemas Cliente/Servidor.

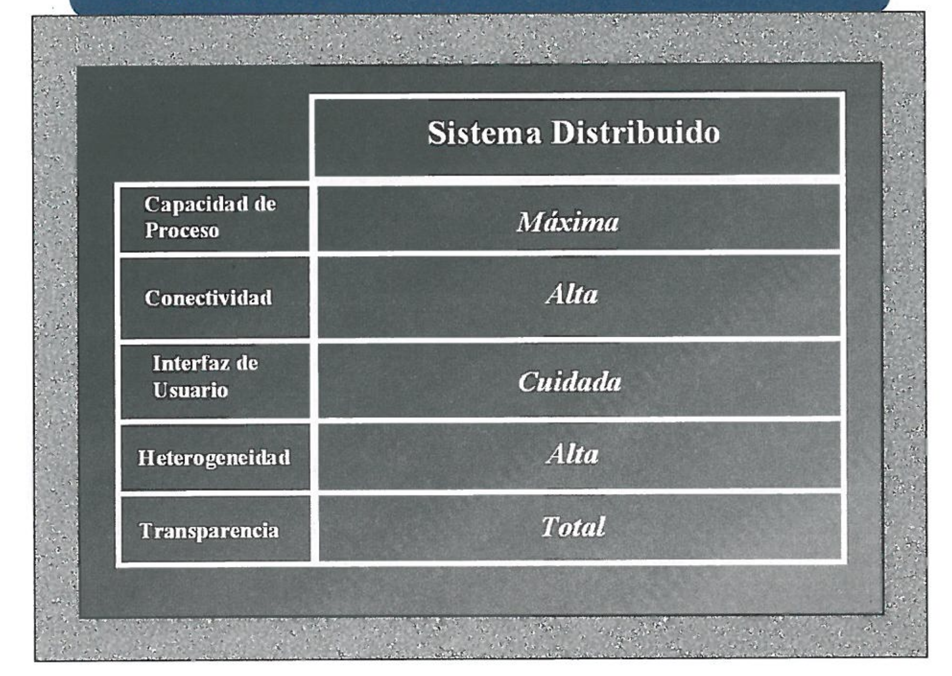
- **Heterogeneidad**: si implementamos nuestro sistema en un lenguaje de programación de amplia difusión - por ejemplo C - y utilizamos librerías presentes en un elevado número de arquitecturas, podremos conseguir que el Sistema Distribuido mantenga las mismas propiedades que las de las Arquitecturas Cliente/Servidor.

- **Capacidad de Proceso**: un Sistema Distribuido va a ser capaz de asignar sus recursos dinámicamente en función de la utilización de las diferentes máquinas. De esta manera podrá descargar ordenadores colapsados hacia máquinas que se encuentren inactivas.

- **Interfaz de Usuario**: aunque no forma parte del concepto de Sistema Distribuido, éste permite implementar diseños modulares en los cuáles la interfaz de usuario puede ser sustituida a medida que las capacidades de las máquinas aumentan.

Además de todas estas ventajas mencionadas, que aúnan las virtudes de los sistemas clásicos, podemos añadir una más, que es propia de los Sistemas Distribuidos y que, por sí sola, sería razón suficiente para sustituir los sistemas clásicos por Sistemas Distribuidos. Esta ventaja es el concepto de *Máquina Virtual Única*.

Figura 2.- Características relevantes de un Sistema Distribuido. Como se puede apreciar, se aprovechan las ventajas de los modelos Cliente/Servidor y Ad-Hoc, introduciéndose además mejoras en puntos donde ambos presentan deficiencias.



	Sistema Distribuido
Capacidad de Proceso	*Máxima*
Conectividad	*Alta*
Interfaz de Usuario	*Cuidada*
Heterogeneidad	*Alta*
Transparencia	*Total*

Máquina Virtual Única: El desafío

Cada vez que configuramos una red nos encontramos con el mismo problema. Asignar direcciones a las máquinas, decidir en qué máquinas vamos a instalar las aplicaciones, elegir los servidores en función de las capacidades de proceso de datos y de almacenamiento, etc...

Esta tarea es compleja y normalmente la lleva a cabo el administrador de sistemas de la empresa, el cual es una persona preparada para ello y con unos conocimientos profundos de lo que está haciendo, así como una gran experiencia en estas tareas.

Sin embargo, el usuario puede no tener esa suerte. Lo más probable es que sea una persona a la cual la frase '¿Cuál es tu dirección IP?' le suene a comentario de Chiquito de la Calzada.

Estas personas, que son mayoría, no deberían tener que saber que su editor favorito está en la máquina tal o cual, que el sistema de archivos que tiene que montar está en tal dirección, ni que para montar su cuenta del servidor tiene que arrancar un programilla que se llama *automount*. De hecho, debería poder acceder a sus datos sin necesidad de saber que existe una *cosa* que se llama servidor.

Además de estas dificultades, los usuarios se encuentran con que la mayoría de las veces el servidor está lleno de gente *trabajando*[1] y la máquina va, como se suele decir en el argot, 'a pedales'. Normalmente los usuarios *listillos* se conectan a otra máquina que esté más descargada para seguir con su trabajo de una manera más cómoda, en términos científicos diríamos que se produce una *redistribución manual* de la carga del sistema.

En un Sistema Distribuido, es el propio *RTS (Run Time System)* o controlador del sistema el que se encarga de distribuir la carga de proceso entre las diferentes máquinas que componen el entorno, de tal manera que no se produzcan cuellos de botella en la capacidad de proceso del conjunto de máquinas de la red.

Finalmente, el *RTS* del sistema será lo suficientemente inteligente como para saber qué máquinas ejecutan mejor una tarea determinada. Si tenemos una red compuesta por un servidor de ficheros, un procesador vectorial y varias estaciones de trabajo, el gestor del sistema tendrá la inteligencia suficiente como para enviar las sumas de matrices al procesador vectorial, dedicar el servidor de ficheros a proporcionar almacenamiento a los usuarios y las estaciones a realizar las tareas de edición.

En definitiva, el concepto de *Máquina Virtual Única* es muy sencillo. Consiste en ocultar al usuario la existencia de una red de ordenadores, mediante la ilusión de que el trabaja en una máquina única y muy grande. Todas las labores de administración de la red, localización de ficheros, etc., se gestiona de manera transparente por el sistema.

De una manera similar al sistema operativo, el cual se encarga de ocultar al usuario - sea este una persona o un programa - las peculiaridades del *hardware* disponible, el *RTS* de un Sistema Distribuido proporciona la ilusión de una *Máquina Virtual Única* ocultando al usuario las peculiaridades de la red.

Objetivos

Tras esta 'pequeña' introducción a las ventajas de los Sistemas Distribuidos, vamos a profundizar en los fundamentos teóricos de los mismos, analizando diferentes conceptos entre los que se incluyen:

- Arquitectura de un Sistema Distribuido.
- Memoria Distribuida.
- Consistencia de Datos.
- Replicación.
- Granularidad.

Desgraciadamente vamos a tener que dedicar todo este artículo - y probablemente parte del siguiente - al análisis teórico de las características de estos sistemas. A partir del segundo artículo elegiremos una arquitectura y realizaremos una implementación de la misma.

Una vez tengamos nuestro Sistema Distribuido funcionando, será el momento de realizar una o dos aplicaciones que sirvan como ejemplo de construcción de aplicaciones distribuidas. Una vez tengamos estas aplicaciones funcionando, y para acabar la serie de artículos sobre el tema, realizaremos una serie de pruebas para comprobar las ventajas de los Sistemas Distribuidos sobre las redes convencionales.

Requisitos

Toda la implementación del gestor del Sistema Distribuido se va a realizar en lenguaje de programación C, sobre el Sistema Operativo *Linux*, sin embargo vamos a intentar mantenerlo lo más estándar posible de tal manera que podamos transportarlo a otras arquitecturas mediante una simple recompilación.

Como nuestro objetivo es didáctico, no necesitamos sacar hasta la última gota de velocidad de nuestra implementación. Por este motivo, todas las comunicaciones entre las distintas máquinas se van a realizar mediante la utilización de *Remote Procedure Calls*. La pérdida en velocidad global del sistema se verá más que compensada con la sencillez de utilización de las mismas en comparación a programar *sockets* UDP/IP directamente.

Las aplicaciones que vamos a construir sobre el sistema serán un poco más particulares. Una de ellas se realizará sobre SVGALIB, debido a lo cual solo podremos utilizarla en *Linux*. El resto podrán utilizarse en cualquier máquina que tenga la librería *curses* instalada.

Fundamentos teóricos

Toda máquina que utiliza más de un microprocesador se puede clasificar dentro de uno de los siguientes dos grupos:

- **Multiprocesadores**: se caracterizan por poseer un único espacio de direcciones, el cual es global y accesible de manera directa a todos los procesadores. Cualquier miembro del sistema puede leer o escribir una palabra en el espacio de direcciones moviendo el dato desde, o hacia, una dirección de memoria determinada. La comunicación se realiza a través de esta memoria compartida.

- **Procesadores Distribuidos**: no poseen un espacio de memoria común siendo, en general, conjuntos de máquinas independientes conectadas mediante redes de mayor o menor velocidad. Todas las comunicaciones entre procesadores se han de efectuar mediante técnicas de paso de mensajes.

En el primero de los casos, el *hardware* es muy complicado de diseñar y construir, sin embargo, las aplicaciones se crean de una manera sencilla ya que el multiproceso del sistema está totalmente oculto al programador.

En el segundo, el *hardware* ya está construido, basta agrupar dos o más PC's o estaciones de trabajo alrededor de una red para tener nuestro Procesador Distribuido. Sin embargo crear aplicaciones es muy complejo, ya que en cada aplicación el programador debe crear su propio sistema de comunicaciones, de equilibro de carga, etc...

El objetivo de nuestro Sistema Distribuido será el de aunar las ventajas de los dos mundos, utilizando el *RTS* del sistema de tal manera que tanto el programador como el usuario puedan abstraerse del concepto de máquinas múltiples y realizar código o utilizar el sistema como si fuera una máquina única.

Procesadores Distribuidos: Arquitectura

Desde el punto de vista del *hardware* del sistema, podemos concebir un sistema de Procesadores Distribuidos como un conjunto de estaciones de trabajo unidos mediante una red. La arquitectura del sis-

Figura 3.- Topología de red en bus. Todos los ordenadores del sistema están conectados por el mismo cable principal.

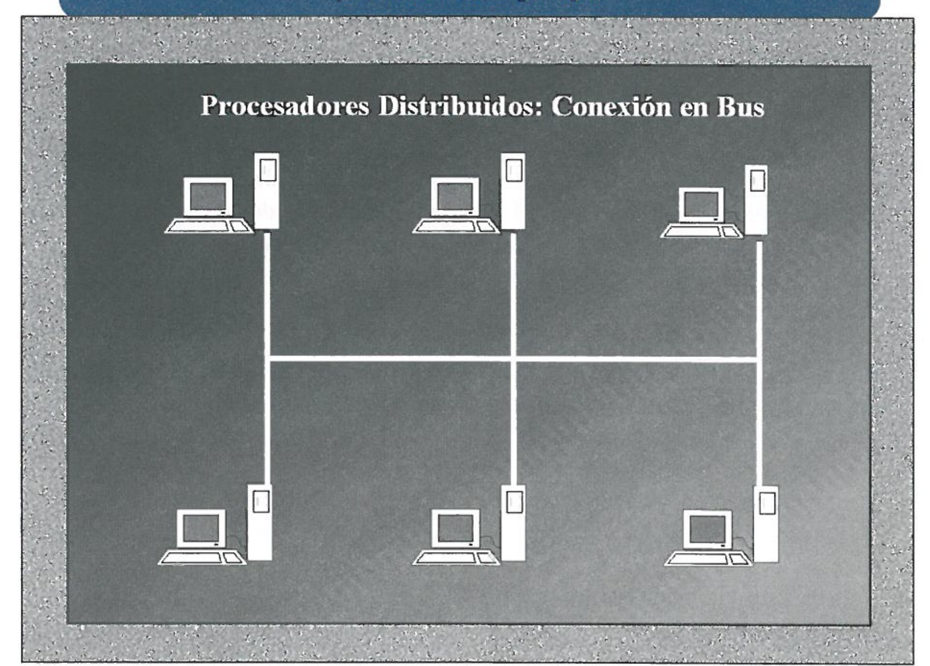
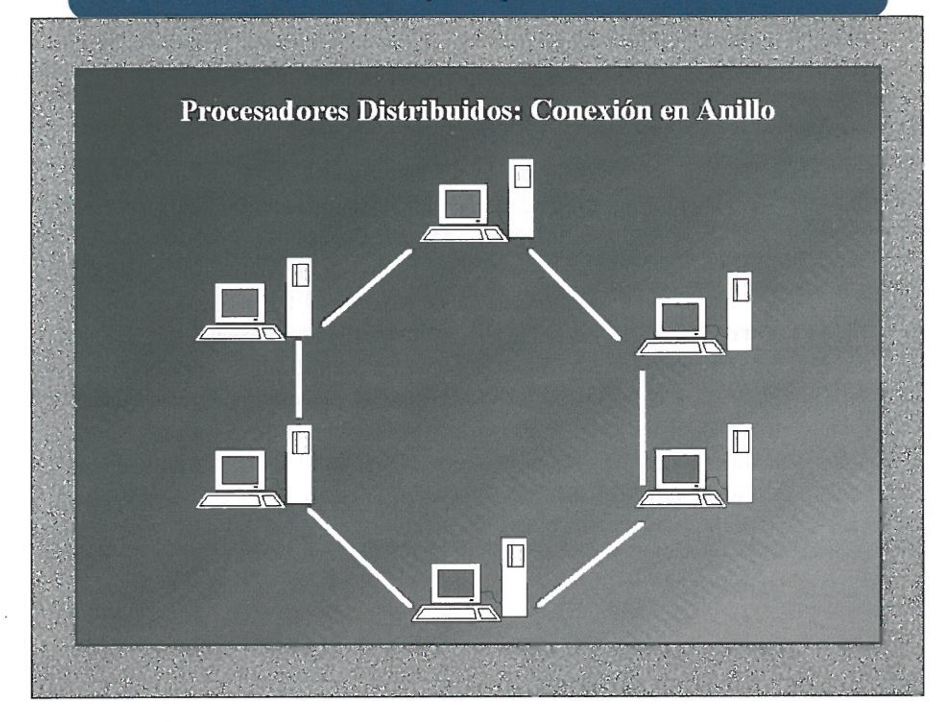


Figura 4.- Topología de red en anillo. Todos los ordenadores del sistema están conectados mediante líneas punto a punto formando un anillo único.



tema dependerá, pues, de la arquitectura de la red que estemos utilizando.

Los tipos de red que se utilizan con más frecuencia en la construcción de Sistemas Distribuidos son:

- *Conexión Directa*: es el método más primitivo y menos flexible, consiste en la interconexión de dos máquinas por medio de una cable.

- *Bus*: surgido como evolución del anterior para poder conectar más de dos máquinas. En esta arquitectura conectamos varios ordenadores a un único cable por donde fluirá toda la información. El problema principal de este sistema es la rápida saturación de la red bien cuando el tráfico de datos es alto o cuando el número de máquinas aumenta indefinidamente. Este problema se puede paliar parcialmente con la incorporación de cachés de acceso en cada una de las máquinas.

- *Anillo*: más complejo de construir y arbitrar que un sistema en bus, el sistema en anillo permite interconectar un número mucho mayor de estaciones y asegura una utilización más completa de la red al eliminar la contención. Su problema principal radica en el aumento de la latencia según se añaden máquinas al anillo.

- *Sistema Conmutado*: los sistemas anteriores funcionan bien siempre y cuando el número de máquinas no sea muy elevado. Si el número de máquinas a conectar es alto, bien los problemas de contención, en el caso del bus, o los de latencia, en el caso del anillo, dejarán el sistema inservible.

  Para paliar este particular problema, se divide el sistema en subredes más pequeñas, interconectadas mediante puntos de conmutación. Este sistema disminuye el tráfico en cada uno de los sectores de la red a cambio de aumentar la complejidad del sistema en un orden de magnitud.

Desde el punto de vista del *software*, debido a que, por definición, los Procesadores Distribuidos no comparten memoria principal, toda comunicación entre ellos debe realizarse mediante técnicas de paso de mensajes.

A lo largo del tiempo se han desarrollado una serie de paradigmas para expresar este paso de mensajes, entre los que destacan:

- *Paradigma SEND/RECEIVE*: en este tipo de sistemas, el más básico de todos, los datos se envían mediante llamadas de envío y recepción. El programador debe ocuparse de las tareas de control de flujo en los envíos, de evitar entrar en una situación inconsistente que bloquee el sistema, etc... En realidad es una avance muy pequeño con respecto a considerar el sistema como ordenadores independientes.

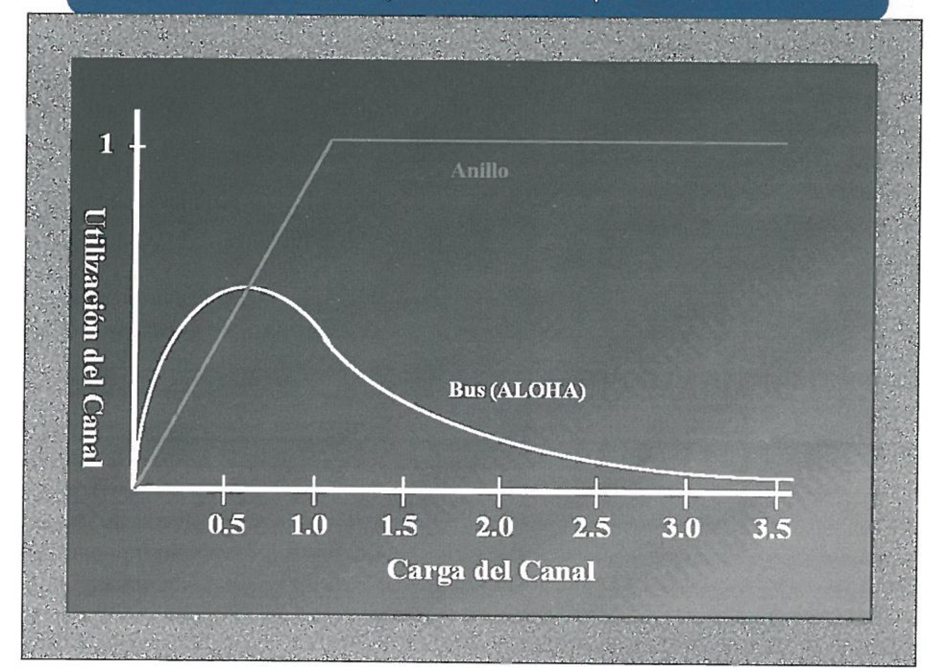
- *Sistemas basados en RPC*: es un paso más allá del sistema anterior. En este caso también enviamos mensajes, sin embargo tareas como control de flujo, recuperación de bloqueos, etc... ya están incorporados en la librería de paso de mensajes.

  Estos sistemas facilitan enormemente la tarea del desarrollador aunque no sean todavía sistemas claramente distribuidos.

- *Memoria Distribuida*: son los verdaderos Sistemas Distribuidos. En ellos se accede a una memoria común a todos los elementos de la red. De proporcionar esta ficción de memoria única y de mantenerla coherente se ocupa el *RTS* o *Run Time System* del Sistema Distribuido.

Por tanto, a la hora de crear nuestro Sistema Distribuido utilizaremos este último método, de tal manera que podamos proporcionar, tanto al programador que utilice nuestro sistema como a su usuario final la necesaria imagen de máquina única.

Figura 5.- Utilización del canal en función de la carga del sistema para redes en bus y en anillo. Como se puede ver, la red en anillo funciona muy bien en condiciones de alta carga, mientras que el bus ofrece un comportamiento mucho peor. En general utilizaremos buses en redes con bajo índice de transferencia de datos y gran número de máquinas y el anillo en el caso opuesto.



Memoria Distribuida: La solución a todos nuestros problemas

Todos los sistemas de memoria compartida poseen la propiedad de que un proceso que se está ejecutando en cualquier máquina puede acceder a los datos residentes en la misma de una manera inmediata. Sin embargo, por muy rápida que sea la red que los comunica, cuando accedemos a un dato que no reside localmente la penalización por el acceso es enorme: hemos de enviar y recibir un mensaje a través de la red.

Como definición podríamos decir que: *Se considera que un sistema implementa un modelo de Memoria Distribuida entre sus miembros si máquinas disjuntas comparten un mismo espacio de direccionamiento*.

En su origen, los sistemas de memoria distribuida dividían la memoria en diferentes páginas, cada una de las cuales residía en un ordenador determinado. Cuando se accedía a una página remota, el sistema provocaba una excepción que se encargaba de traer la página. Este sistema es muy sencillo, pero su rendimiento es abismalmente pobre si tenemos un porcentaje muy alto de fallos de página.

Para resolver este problema se han propuesto diferentes modificaciones al modelo de Memoria Distribuida, entre las que destacan:

- La debilitación de la semántica de la Memoria Distribuida. Originado en los Sistemas Multiprocesadores, se basa en imponer ciertas condiciones a la memoria de tal manera que el valor que nos devuelve una lectura ya no tiene por qué ser el que se ha escrito más recientemente en ella. El precio a pagar por el incremento en la velocidad es doble: por un lado trasladamos complejidad al usuario y por el otro hacemos que cierto tipo de errores sean muy difíciles de trazar.
- Basar la compartición de memoria en estructuras de más alto nivel, las cuales estén más adecuadas a los

Figura 6.- Paradigma de Memoria Distribuida. Todas las máquinas comparten un espacio de memoria único, el cual es transparente para el programador y el usuario.Desde cualquier punto de vista, los datos están en una nube que envuelve al sistema.

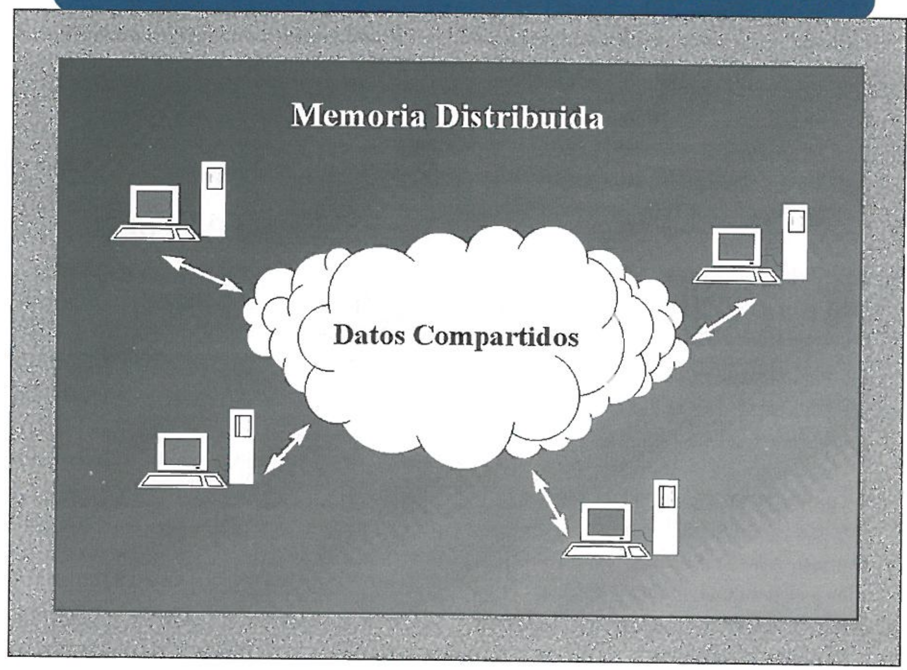
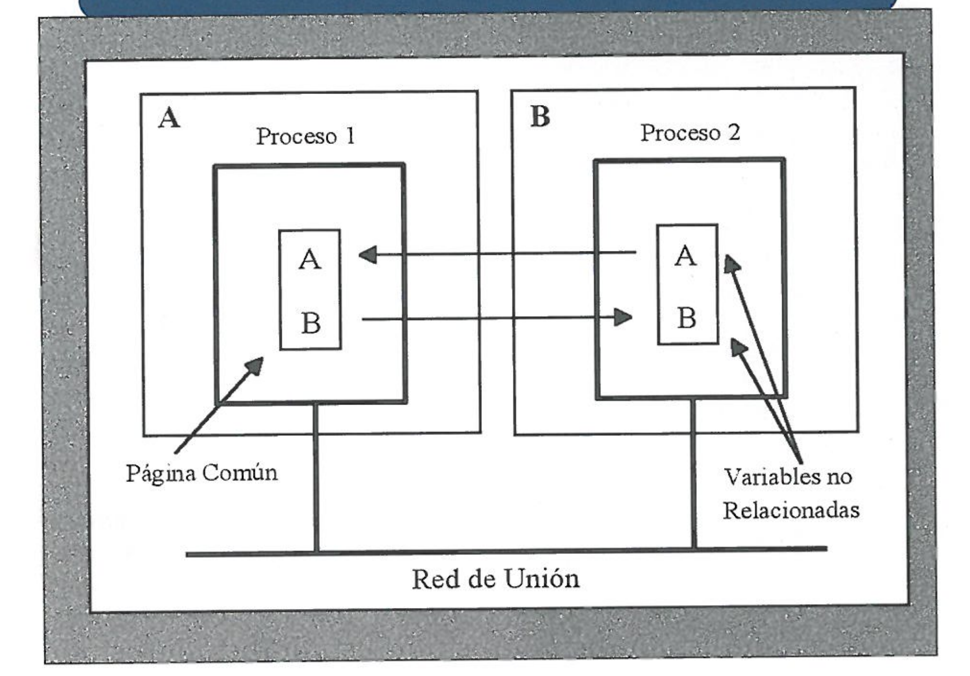


Figura 7.- Uno de los principales problemas de rendimiento del los sistemas basados en Páginas Distribuidas es la llamada falsa compartición de datos. En ella dos variables independientes, las cuales residen en la misma página, son utilizadas por dos procesos que ejecutan en máquinas distintas. El resultado es que la página viaja constantemente por la red, ralentizando en varios órdenes de magnitud la ejecución de ambos procesos.



problemas que se quieran resolver. En este caso el precio a pagar es una pérdida en la generalidad de la información representada.

- Dividir el espacio de memoria en dos partes, una compartida y otra que no lo está, de tal manera que la mayor parte de las operaciones que se realicen tengan lugar en memoria local. En estos casos el programador ha de tener muy claro qué datos pertenecen a cada parte para así optimizar el rendimiento de su aplicación.

- La introducción de *cachés* de datos, en las cuales guardamos valores accedidos recientemente. Basándonos en el principio de localidad espacial y temporal de referencia de datos, podremos esperar que todos aquellos valores a los que hemos accedido recientemente van a volver a ser accedidos en un futuro. En este caso disminuimos el número de mensajes que circulan a través de la red a cambio de implementar un sistema mucho más complejo.

Como podemos ver, todas las mejoras propuestas se basan, bien en trasladar parte de la complejidad del sistema al desarrollador de aplicaciones, o bien en utilizar a priori información sobre el tipo de aplicaciones que van a correr en el sistema, con lo cual obtenemos una pérdida de generalidad en el mismo.

Modelos de Consistencia

Si además de tener páginas o variables compartidas que circulen a través de la red, colocamos una pequeña caché de datos en cada uno de los miembros del Sistema Distribuido, conseguimos resultados espectaculares en el rendimiento global del entorno. El motivo de esta mejora es el aumento radical en la proporción entre accesos locales y remotos.

El precio a pagar por este comportamiento es, como ya se ha dicho, un incremento en la complejidad del Sistema Distribuido, ya que necesitamos algún mecanismo que asegure en todo momento la **consistencia** del conjunto de la memoria.

Mantener copias de los datos en diferentes máquinas de la red exige el envío de numerosos mensajes de control a fin de que las copias estén actualizadas en todo momento. Este comportamiento hace que, de nuevo, el rendimiento del sistema disminuya. Para evitar este problema, relajamos las condiciones de consistencia, realizando una serie de suposiciones que contribuyen al aumento de la velocidad final.

De esta manera, lo que se denomina *modelo de consistencia*, no es más que un pliego de condiciones en el que se detallan las normas que ha de cumplir el programador para que el sistema de memoria se comporte de la manera que lo hace en un ordenador convencional.

Los distintos modelos existentes se pueden clasificar en dos grupos:

- Modelos de Consistencia con Operaciones de Sincronización.
- Modelos de Consistencia sin Operaciones de Sincronización.

Modelos sin Operaciones de Sincronización

Se caracterizan porque la sincronización entre las distintas copias de los datos se hacen de manera transparente al programador, el cual simplemente ha de atenerse a las condiciones impuestas en el modelo.

Los distintos modelos son, en orden creciente de eficiencia:

- **Consistencia Estricta:** simula el comportamiento clásico de las

máquinas de un único procesador. Todas las escrituras realizadas en la memoria compartida son instantáneamente reflejadas en todas las copias, manteniéndose un orden temporal absoluto.

- **Consistencia Secuencial:** un sistema posee este tipo de consistencia cuando las operaciones realizadas en un mismo procesador cumplen un orden estricto, aunque las que se realicen en distintos procesadores cumplan un orden arbitrario. Este tipo de sistemas garantizan que todos los procesos ven los accesos a memoria en la misma secuencia, aunque no garantiza que ésta se encuentre en un orden temporal exacto.

- **Casual:** distingue dos tipos de datos: los que están relacionados potencialmente y los que no lo están. A partir de esta premisa, podemos relajar las condiciones del modelo, de tal manera que solo las operaciones relacionadas han de cumplir un orden temporal exacto. Las demás pueden estar ordenadas arbitrariamente.

- **PRAM:** relaja aun más la consistencia *casual*. En un sistema *PRAM*, basta con que las escrituras realizadas por un proceso sean vistas en el mismo orden por todos los demás procesos. Escrituras realizadas por diferentes procesos pueden ser vistas en distintos órdenes por los distintos procesos. A diferencia de los sistemas anteriores, a partir de la consistencia *PRAM*, cada proceso ve un orden temporal distinto.

- **de Procesador:** levemente más lenta que la consistencia *PRAM*, es muy similar a ésta introduciendo la restricción de que los accesos realizados a la misma posición de memoria deben ser vistos en igual orden por todos los procesos. Esta condición se denomina *coherencia de memoria*.

Modelos con Operaciones de Sincronización

Son menos importante para nosotros, ya que introducen una serie de condiciones que hacen desaparecer el concepto de *Máquina Virtual Única*. Se basan en la utilización de zonas críticas en las cuales aparecen las variables de sincronización, las cuales limitan el acceso a los datos hasta que se haya abandonado la zona crítica del código.

Al utilizar este sistema, solo es necesario propagar a las demás máquinas los valores que se obtienen a la salida de la región crítica, reduciendo el número de transacciones.

Los tipos más comunes son:

- Consistencia Débil.
- Consistencia de Liberación.
- Consistencia de Entrada.

Otras Consideraciones

Además de los conceptos que hemos ilustrado, existen otra serie de aspectos relevantes a la hora de diseñar un Sistema Distribuido:

- La **Granularidad** del sistema es el tamaño mínimo de información que puede viajar entre máquinas de una manera independiente. Podemos trasladar una página de memoria, una variable, un grupo de variables, etc... Granularidades grandes disminuyen el número de transacciones aumentando el rendimiento del sistema. Sin embargo pueden crear problemas de vapuleo por *falsa compartición* (ver Figura 7).

- La estrategia de **Replicación** es uno de los factores que más afecta el rendimiento de un Sistema Distribuido. Una estrategia adaptada al tipo de aplicaciones a ejecutar puede suponer una gran diferencia, de hasta un orden de magnitud, en la velocidad.

Conclusiones

El desarrollo de un Sistema Distribuido es enormemente complejo desde el punto de vista de la implementación. Sin embargo esa complejidad no es nada en comparación a las dificultades que podemos encontrar si no tenemos una idea clara de lo que se quiere hacer, por qué se debe hacer y como debemos hacerlo.

Éste, y no otro, es el motivo por el que nos hemos extendido tanto en los fundamentos teóricos que hay detrás de la noción de Sistema Distribuido. Conceptos como *página distribuida, Procesadores Distribuidos, Modelos de Consistencia,* etc.., deben ser algo que tengamos muy claro antes de escribir una sola línea de código, si es que queremos que nuestro barco - distribuido, por supuesto - llegue a buen puerto.

En el próximo artículo describiremos la arquitectura del sistema que vamos a implementar y comenzaremos a escribir el código necesario para crearlo.

Jorge Delgado Mendoza es miembro de XFree86 desde el año 1994, como desarrollador responsable, entre otras cosas, del servidor de Oak Technologies Inc. Es Ingeniero Superior de Telecomunicaciones por la UPM y trabaja actualmente en Convex SuperComputers. Su correo electrónico es **ernar@convex.es.**

[1-2] La cursiva es totalmente intencional. La mayor parte de los trabajos de edición, preparación de presentaciones, etc..., apenas ocupan CPU. Sin embargo últimamente las máquinas se colapsan rápidamente. Un análisis de los procesos que se están ejecutando suele dar como resultado que valores de hasta el 80% de la CPU se ocupan en descomprimir y presentar los JPG's y GIF's de *Internet*.

DELPHI

Sockets y programación Internet con Delphi

Enrique de la Lastra

Programar aplicaciones para Internet es un trabajo para el que hay que dominar conocimientos de diversos ámbitos: no sólo es necesario conocer los protocolos de comunicación y de aplicación de Internet, sino saber cómo implementar estos conocimientos.

Introducción

Internet ha sido y está siendo el "boom" en España y en casi todo el mundo debido a la introducción de la tecnología *World Wide Web* (WWW) más conocida como "el Web". Hasta la fecha en que Tim Bernes Lee creó el lenguaje de hipertexto HTML (*HyperText Markup Language*), esto es 1989, el ámbito de Internet se ceñía a las Universidades y a unas pocas empresas y su utilización se reducía casi exclusivamente al envío de correo electrónico y a la transferencia de ficheros mediante FTP (lo que no era poco). Además algunos, especialmente en las Universidades, aprovechaban la red para conectarse a máquinas potentes que se encontraban a muchos kilómetros de sus lugares de trabajo, para realizar allí el procesamiento de datos.

Pero la aparición del lenguaje HTML, del protocolo HTTP (*HyperText Transfer Protocol*, Protocolo de Transferencia de Hipertexto), y la creación de los *"browsers"* (hojeadores), es decir, de herramientas capaces de interpretar el formato HTML, extendieron el uso de Internet hasta el punto en que lo conocemos actualmente. El porqué de esta *revolución* es sencillo: hasta el HTML, la información se distribuía como texto plano y para acceder a ella había que introducir comandos de texto con molestos parámetros difíciles de recordar. A partir del HTML, la información se presenta con formato hipermedia (texto, gráficos, imágenes, sonidos y vídeo conjuntados en un documento) y el acceso a dicha información se realiza seleccionando en un entorno gráfico, mediante el puntero del ratón, un enlace a otro documento.

Como la tecnología en todos los ámbitos evoluciona a un ritmo vertiginoso, Internet no podía verse ajena a esta evolución: CGI, Java, JavaScript, Visual Basic Script, VRML, ActiveX, CORBA y un largo etcétera de nuevas tecnologías han nacido al amparo de Internet en los dos últimos años. La herramienta que nos ocupa, Delphi, también ha evolucionado para adaptarse a los nuevos tiempos, incluyendo numerosas facilidades para la programación Web.

En este artículo veremos las herramientas de desarrollo para Internet que se han añadido en la nueva edición de Delphi (Delphi 3.0) y haremos una introducción al desarrollo de programas para la Internet y el Web mediante el compilador Delphi (versiones 1.0, 2.0 y 3.0).

Facilidades Internet en Delphi 3.0

En la edición 3.0 de Delphi se han añadido una serie de componentes que facilitan la creación de programas para Internet.

Es importante señalar que estos componentes sólo se incluyen en las versiones "Profesional" y "Cliente/Servidor", y no en la versión "Estándar" (que es la barata). Si ya ha comprado la versión estándar y está pensando desarrollar aplicaciones Web, debería actualizarse a una de las versiones superiores o proponerse realizar un gran trabajo para implementar cualquiera de los componentes para Internet.

En las versiones "Profesional" y "Cliente/Servidor" del paquete se incluyen los siguientes componentes para aplicaciones Web:

1. **ActiveForms**
2. **Internet Solutions Pack**

Sólo la versión "Cliente/Servidor" incluye además otras herramientas útiles para la creación de servidores Web (**WebServer,** para crear aplicaciones de datos de alta velocidad y productividad; **WebBridge,** que ofrece una API común para dar soporte de NSAPI - API del Servidor de Netscape - e ISAPI - API del Servidor de Microsoft -; **WebModules, WebDispatcher,**...)

Nos detenemos a estudiar cada uno de ellos con mayor detalle.

Active Forms de un sólo paso

Como ya explicamos en el artículo del número anterior dedicado a Delphi 3.0, para crear un ActiveForm sólo tenemos que hacer uso de un "Experto" con el que es posible convertir cualquier formulario (*"Form"*) normal (con sus botones, cajas de texto, etc., así como con el código de manejo de eventos...) en un "Formulario Activo", es decir, un *ActiveX* que contiene los controles que hemos introducido en nuestro formulario.

Dos son las aplicaciones prácticas más importantes de los *"Active Forms".* Por un lado, se facilita la integración con otros entornos de desarrollo. Pero la aplicación más importante es la de crear aplicaciones distribuibles en Internet o la de crear clientes ultra-delgados en una arquitectura de múltiple nivel (tecnología Multi-Tier, que se explicó también en el número anterior).

Internet Solutions Pack

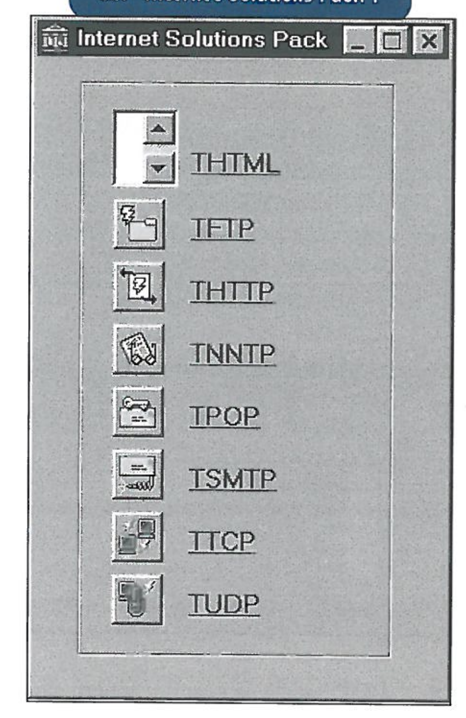
El *"Internet Solutions Pack"* es un conjunto de ocho componentes que se agrupan en la pestaña "Internet" de la paleta de componentes de Delphi 3.0, y que implementan los más importantes protocolos de Internet para aplicaciones cliente. Estos componentes están implementados como controles ActiveX. Se muestran en la Figura 1, colocados sobre un formulario de Delphi.

El "Internet Solutions Pack" son ocho componentes ActiveX que implementan los protocolos cliente más comunes de Internet: FTP, HTTP, POP, SMTP,...

Conozcamos a continuación para qué sirve cada uno.

1. **HTML:** visor (o navegador, o *browser*) HTML con opciones de recuperación automática de documentos HTML desde Internet y análisis de datos HTML. También, puede utilizarse como analizador o procesador no visible de documentos HTML.

Figura 1: Componentes de Internet del "Internet Solutions Pack".



2. **FTP:** cliente FTP (*File Transfer Protocol*) para transferencia de archivos desde y hacia un servidor FTP.

3. **HTTP:** cliente HTTP que permite recuperar documentos HTML si no es necesario el procesamiento de la página o de las imágenes.

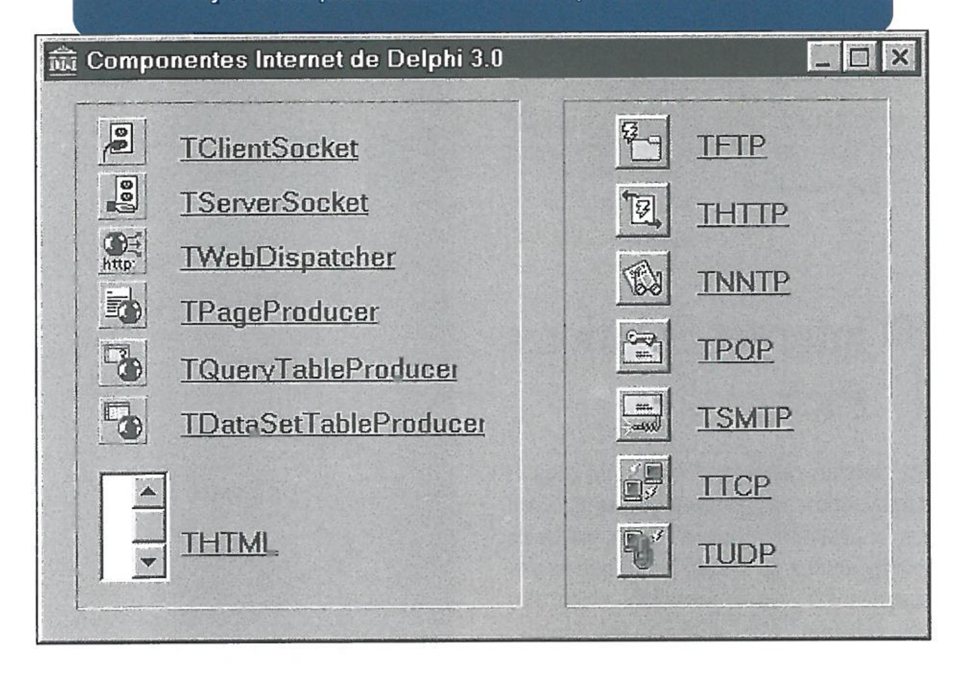
4. **NNTP:** cliente NNTP (*Networking News Transfer Protocol*) que ofrece capacidades de lectura y envío de mensajes.

5. **POP:** cliente POP3 (*Post Office* Protocol) para recuperar e-mail desde servidores que soporten este protocolo.

6. **SMTP:** ofrece la posibilidad de acceder a servidores de e-mail SMTP (*Simple E-mail Transfer Protocol*) y de enviar correo.

7. **TCP:** implementa el protocolo WinSock TCP (*Transmission Control Protocol*) tanto para aplicaciones cliente como servidoras, facilitando el acceso a los servicios del protocolo TCP.

Figura 2: Componentes de Internet de Delphi 3.0 Client/Server.



8. **UDP**: igual que el anterior pero para el protocolo UDP (*Data Protocol*)

Herramientas Y facilidades de la versión Cliente/Servidor

En la versión Cliente/Servidor de Delphi 3.0, se incluyen, además de los componentes anteriores, facilidades para la creación de sedes Web y, por tanto, para la creación de aplicaciones Internet e Intranet. Los componentes de esta versión, que se agrupan también en la pestaña "Internet" de la paleta de componentes, se muestran en la Figura 2. Estos componentes se encargan de toda la comunicación con el servidor de manera que, el programador sólo debe preocuparse del contenido de la sede Web y no de los protocolos de comunicaciones HTTP.

1. **WebServer**: Con Delphi 3.0 se pueden aprovechar los conocimientos en el desarrollo de software Cliente/Servidor para crear aplicaciones Web. Las aplicaciones WebServer son DLL's residentes en el servidor Web y que se asocian directamente con ISAPI y NSAPI. Con los programas WebServer, se dispone de un control total sobre una página Web sin preocuparse de la gestión de recursos y la gestión de estados. Para crear una aplicación WebServer hay que seleccionar la opción "New..." en el menú "File", y, posteriormente, pulsar el icono "Web Server Application" (ver.Figura 3).

Las aplicaciones WebServer son DLL's que residen en el servidor Web y se asocian directamente con ISAPI y NSAPI.

2. **WebBridge: WebBridge ofrece a los desarrolladores una API común para programar con** NSAPI e ISAPI, lo que evita tener que reescribir código si dichas API's propietarias cambian. Es decir, *WebBridge* hace de puente del servidor Web que se esté utilizando en cada momento.

3. **WebDispatcher: El componente WebDispatcher es un componente cuya misión es el control de los** sucesos en el Web. Actúa de forma sincronizada con todos los componentes de consulta para producir aplicaciones CGI y HTML, permitiendo al programador producir contenidos Web mediante los mismos mecanismos que son utilizados en el desarrollo de aplicaciones Cliente/Servidor.

4. **ClientSocket**: Componente para convertir la aplicación en un cliente TCP/IP. Se encarga de todo el proceso de apertura, mantenimiento y finalización de la conexión con el servidor TCP/IP.

5. **ServerSocket**: Componente que convierte la aplicación en un servidor TCP/IP. Escucha las diferentes peticiones de conexiones TCP/IP de otras máquinas y establece las mencionadas conexiones. Este componente y el anterior se encargan de manejar los protocolos de comunicación de bajo nivel (TCP, *Transmission Control Protocol* e IP, *Internet Protocol*) para facilitar el desarrollo de aplicaciones de más alto nivel.

6. **PageProducer**: Componente para convertir una plantilla HTML en una cadena de comandos HTML que pueden ser interpretados por cualquier navegador.

7. **DataSetTableProducer y QueryTableProducer**: Componentes que unen una secuencia de comandos de HTML para generar una salida en formato tabla, de los registros de un TQuery o un TDataSet (que obtendrán los parámetros de un mensaje HTTP de petición o de respuesta respectivamente).

Desarrollo de aplicaciones Internet

Quienes no dispongan de Delphi 3.0 y, además, no piensen actualizarse, al menos a corto plazo, tienen la posibilidad de desarrollar sus propias aplicaciones Internet partiendo de cero.

Para empezar a programar aplicaciones para Internet, hay que conocer qué son los *Windows Sockets* y no hay otra manera de explicarlo sino comenzando por la estructura de protocolos de la red Internet.

Windows Sockets

Internet se basa en una serie de protocolos de comunicación que se organizan en varias "capas" que van montadas una encima de otra, al igual que las capas de una cebolla. Cada "capa" ofrece unas funciones (o un servicio) para la capa inmediatamente superior a la vez que oculta la implementación de dichas funciones. De esta forma, una capa utiliza el servicio de su capa inmediatamente inferior y ofrece un nuevo servicio a la superior. Cada uno de los protocolos de Internet actúa en sólo una de las capas.

El protocolo TCP/IP es, en realidad, una conjunción de dos protocolos: el TCP y el IP. Debajo de los protocolos TCP/IP hay dos capas: una que controla el medio de transmisión (el cable) y otra que controla el tipo de red (que generalmente será una Ethernet). El protocolo IP está inmediatamente encima, en la capa 3 mientras que, en la capa 4, se encuentra el protocolo TCP. Estos dos protocolos son necesarios para establecer, mantener y liberar una comunicación fiable entre dos ordenadores que no se encuentran en la misma red. Sin los protocolos TCP/IP no habría comunicación más allá de un entorno local. Sin entrar en detalle en los mecanismos empleados por estos protocolos, nos basta saber que sin ellos no hay comunicación posible en Internet ni en entornos Intranet.

Una vez llegados aquí, ya podemos definir los *sockets*: son la implementación en software de los protocolos TCP/IP. Los *sockets* evitan que tengamos que conocer los detalles de funcionamiento de estos protocolos. Simplemente debemos utilizar las funciones que nos ofrecen y despreocuparnos de lo que ocurre debajo. Por tanto, los *sockets* son la interfaz entre las aplicaciones de Internet y el Software de red interno.

Los sockets representan la interfaz entre las aplicaciones de alto nivel de Internet y el software de red interno (TCP/IP).

Los *Windows Sockets* o "WINSOCK", como ya habrá adivinado el lector, son los *sockets* para el entorno Windows.

Los *sockets* originales fueron desarrollados para el Unix de Berkeley. Sin embargo, como Windows no tiene todas las características de los sistemas operativos avanzados ya que, sólo maneja eventos (como clics del ratón), los *sockets* se adaptaron a Windows para que integrasen eventos de la red.

Cualquier programador puede crear programas de aplicación utilizando la interfaz WINSOCK: sistemas e-mail, *browsers*, utilidades de transferencia de ficheros..., pueden ser construidas sobre el WINSOCK.

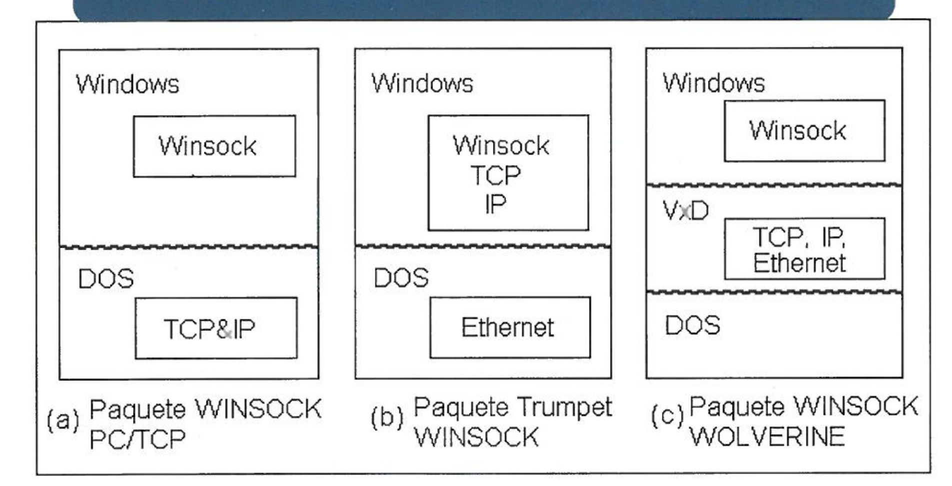
Antes, había que desarrollar ambos: tanto los programas de aplicación como los programas de comunicaciones

La interfaz WINSOCK se implementa como una librería DLL (*Dynamic Link Library*, Librería de Enlace Dinámico), que tiene el mismo nombre: WINSOCK.DLL.

Existen distintas versiones de esta librería que suponen una evolución desde la primera que vio la luz. Las analizamos en el siguiente apartado.

Figura 3: Creación de una aplicación WebServer.

Figura 4: Estructura de la librería WINSOCK.DLL según las versiones.



Versiones de Winsock.Dll

La primera generación de productos TCP/IP fue diseñada para soportar programas en DOS. El soporte del adaptador Ethernet y de los protocolos TCP/IP residían en el área de 640K del DOS (absorbiendo 100K de esa memoria). Cuando Windows empezó a hacerse popular, se implementó un pequeño WINSOCK.DLL que permitía a los programas Windows utilizar el código que residía en la memoria del DOS. El antiguo paquete PC/TCP se diseñó de este modo (Figura 4 -a-). En dicho paquete, la interfaz de red se compartía entre los programas de Windows y los de DOS.

Cuando Windows se estandarizó se desarrollaron *sockets* que residían más allá de los 640K del DOS y de los cuáles sólo una pequeña parte residía en la memoria DOS (la parte que soportaba el adaptador de red). Netmanage Chameleon y el popular Trumpet WINSOCK se construyeron de esta manera (Figura 4 -b-).

Una opción algo más sofisticada para Windows 3.1 y Windows 3.11, es empaquetar el soporte de algo llamado VxD. De igual forma que los módulos DLL, un VxD reside más allá de los 640K del DOS. Sin embargo, un VxD es parte del supervisor del sistema Windows y tiene más control de la memoria y de los dispositivos reales. El nuevo soporte de Internet, llamado "Wolverine" está escrito de esta forma y es gratis en la actualización Win 3.11 (Figura 4 -c-).

En sistemas operativos avanzados (OS/2 ó Windows NT), las aplicaciones de 16 bits corren en una "máquina virtual". Así, esas aplicaciones "ven" un PC estándar corriendo en Windows 3.1 puro. En este entorno, WINSOCK.DLL es una rutina que transfiere todas las peticiones al S.O. externo. Los antiguos programas Windows comparten la red con los otros programas multitarea que corren bajo OS/2 o NT (Figura 5).

En Windows 95, el software Winsock se instala con el sistema operativo, en dos ficheros: WINSOCK.DLL (versión 16 bits) y WSOCK32.DLL (versión 32 bits)

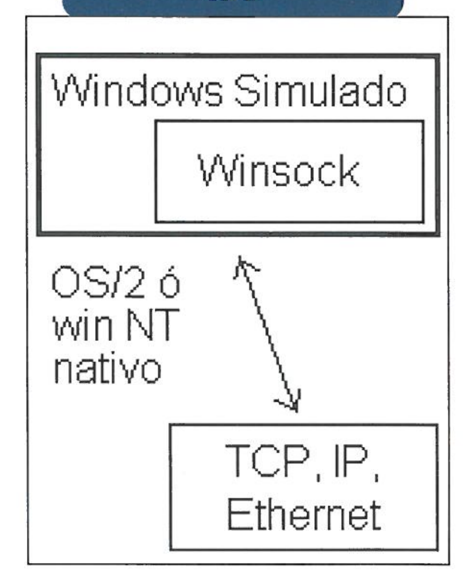
En Windows 95, el software Winsock está incorporado en la instalación del sistema operativo, en dos ficheros: WINSOCK.DLL (versión 16 bits) y WSOCK32.DLL (versión 32 bits). Al ser Windows 95 un sistema operativo de 32 bits y de memoria lineal, no es necesario realizar ningún "apaño" para llamar a esta librería, ya que no tiene las limitaciones de memoria del MS-DOS. Los programas de 16 bits simplemente harán uso de la librería WINSOCK.DLL, mientras los de 32 bits lo harán de WSOCK32.DLL.

Es importante destacar que al instalarse con el sistema operativo, en Windows 95 (como ocurre en UNIX desde sus primeras implementaciones) no es necesario añadir ningún soporte para la programación TCP/IP. Sin embargo, si trabajamos con Windows 3.11 (y por tanto con Delphi 1.0, si queremos programar algo) sí hay que instalar el soporte para TCP/IP. Es muy recomendable el software "Wolverine", ya que es gratuito para los usuarios registrados de Windows 3.11 (para trabajo en grupo). Se puede "bajar" de Internet mediante *ftp* en:

ftp://ftp.microsoft.com/peropsys/windows/public/tcpip/wfw32.exe

El fichero wfw32.exe es auto-descomprimible; por lo tanto habrá que situarlo en el directorio donde queramos almacenar el software de red Wolverine y ejecutarlo.

Figura 5: Funcionamiento de WINSOCK.DLL en Windows NT y OS/2.



Y alguno podría preguntarse: ¿y cómo es posible bajarse algo de Internet, es decir, estar conectado a Internet, sin tener el software TCP/IP instalado en el ordenador?. La respuesta es: gracias al protocolo PPP (Point to Point Protocol), que es un protocolo punto a punto que nos permite conectarnos vía telefónica al proveedor de servicios de información. El protocolo PPP es una implementación completa del TCP/IP, pero que sólo funciona en conexiones punto a punto (esto es, sólo permite conectar un ordenador a otro a través de un enlace directo). No sirve como protocolo TCP/IP en redes de ordenadores que comparten un mismo medio físico.

El protocolo PPP es una versión mejorada de otro protocolo punto a punto: el SLIP. Ambos proveen la misma funcionalidad pero el primero es más potente. Hay que destacar que el software Wolverine no implementa los protocolos SLIP ni PPP (en el Web de Microsoft dicen que "está en desarrollo" para una futura versión, pero que en cualquier caso se incorporará en la nueva edición de Windows y de Windows NT. Esperemos que así sea).

¿Pero qué es un socket?

Ya hemos visto lo que son los *sockets*, pero desde el punto de vista del programador que va a hacer uso de ellos: representan la interfaz del protocolo TCP/IP para las aplicaciones de red. Sin embargo, técnicamente un *socket* difiere de este concepto.

El protocolo IP es el encargado de encaminar la información al destino. Para ello se utiliza una dirección para cada ordenador, única en todo el mundo. Esta dirección está formada por 32 bits y se representa en grupos de 8 bits separados por puntos. En forma decimal una dirección IP tiene el siguiente aspecto: 123.123.123.123 (por ejemplo: 191.132.1.24). Toda vez que un paquete de datos llega al destino gracias al encaminamiento que proporciona el protocolo IP, debe conocer a qué aplicación (o servicio de red) se dirige. Para ello se emplea un número de puerto de 16 bits, que indica el proceso de aplicación que debe recibir esos datos.

Con una dirección IP y un número de puerto se define unívocamente una aplicación concreta de un ordenador determinado. Cada pareja DIRECCIÓN IP + NÚMERO DE PUERTO, define un servicio de red único en el mundo. A esta combinación de dirección y puerto es a lo que denominamos *socket*.

Un socket es en realidad una pareja de datos: una dirección IP (que define un ordenador único) y un puerto TCP (que define un servicio de red).

Existen una serie de puertos asignados de manera fija a unas cuantas aplicaciones comunes y estandarizadas de Internet. En la tabla 1 se muestran algunos de éstos.

Los puertos 1 a 255 se reservan para las aplicaciones más comunes. Del 256 al 1023 son utilizados por servicios de máquinas UNIX. Por último los puertos 1024 al 65535 son los llamados "puertos reservados", la mayoría de los cuales se emplean para asignaciones dinámicas. Las asignaciones dinámicas de puertos tienen la ventaja de que la aplicación de red no necesita usar ningún puerto específico y por tanto habilitan al cliente para conectarse con varios servidores al mismo tiempo. Esto es lo que permite que podamos abrir desde un PC y al mismo tiempo, varias conexiones con un navegador a un mismo o a distintos sitios Web (el cliente genera aleatoriamente su propio número de puerto para cada petición que envía a un servidor Web, mientras que utiliza un número fijo para el puerto del servidor Web, es decir el puerto 80. Las respuestas de cada servidor llevarán como puerto propio el 80 y como puerto de cliente el que éste generó).

Tabla 1: Algunos de los puertos TCP reservados para las aplicaciones comunes de Internet.

fila 0	col1	col2
fila 1	protocolo	puerto
fila 2	ftp	21
fila 3	telnet	23
fila 4	smtp	25
fila 5	http	80
fila 6	pop	109
fila 7	pop3	110
fila 8	nntp	119
fila 9	snmp	161
fila 10	login	513
fila 11	talk	517

Más información sobre Sockets

En el CD-ROM de la revista se incluye la especificación completa de los *Windows Sockets*, versión 1.1 y versión 2.0, en formato de texto plano (Wsock11.txt y Wsock20.txt), y en formato MS Word 6.0 (Wsock11.doc y Wsock20.doc); también se incluye el fichero de cabecera de la librería WINSOCK.DLL en C (winsock.h) y en Pascal (winsock.pas e winsock.inc). La dirección en el Web de Microsoft donde puede encontrar documentos de las versiones 1.1 y 2.0 en varios formatos son, respectivamente:

ftp://ftp.microsoft.com/bussys/winsock/spec11/
ftp://ftp.microsoft.com/bussys/winsock/winsock2/

Si se desea obtener más información acerca de la API de los *Windows Sockets*

se puede buscar, mediante FTP anónimo en la dirección ftp:

ftp://sunsite.unc.edu/pub/micro/pc-stuff/ms-windows/winsock

Allí se puede encontrar una gran variedad de información acerca de los *Windows Sockets* no sólo de Microsoft, además de especificaciones, aplicaciones, DLL's de prueba y guías de usuario. En concreto, existe un archivo (FAQ) con una lista de las preguntas más corrientes sobre la interfaz Winsock, junto con sus correspondientes respuestas. En el directorio **apps/** se pueden encontrar aplicaciones que corren sobre la interfaz de aplicación de WINSOCK.DLL y en el directorio **packages/** están situadas varias librerías WINSOCK.DLL de distintos fabricantes.

¿Cómo accedemos a los Sockets?

Ahora ya hemos comprendido qué son los *sockets* y para qué sirven. Ahora queda por resolver la parte más complicada: ¿cómo podemos acceder a las diferentes funciones que ofrecen los *sockets* (en concreto Winsock)? En la librería WINSOCK.DLL se definen una serie de funciones, estructuras, constantes y variables, a las que resulta fácil acceder desde ficheros de C.

Hasta la versión 2.0 de Delphi teníamos dos opciones cuando queríamos programar con *sockets*: la primera, utilizar directamente las funciones tal cual se proporcionan en el Winsock; la segunda, crear una *unit* en Delphi que *encapsulase* toda la funcionalidad requerida de los Winsock (que no tiene porqué ser todo lo que ofrecen los *sockets*, sino sólo aquello que nos interesa) de forma que acceda a las funciones de dicha *unit* en lugar de acceder directamente a las funciones de WINSOCK.DLL.

La primera opción pudiera parecer más rápida en un principio, pero al igual que cuando utilizamos en Delphi funciones de la API de Windows, el trabajo es mucho más tedioso ya que hay que tratar con parámetros "extraños" en lugar de utilizar una interfaz con parámetros en Pascal. De hecho, el propio compilador Delphi no es más que una *encapsulación* "agradable" y fácil de usar de la API subyacente del Windows. El mero hecho de crear un botón en Windows, es una labor tediosa para la que hay que reservar memoria, asignar un *handle* a la ventana, pintar el botón,... Delphi soslaya todo este proceso para que al programador le resulte fácil la creación y destrucción de un botón, o de cualquier otra ventana. Así pues, la segunda opción se presenta como la mejor.

A partir de Delphi 2.0, como éste es un compilador de 32 bits y sólo funciona con sistemas operativos de 32 bits (a partir de Windows 95, que como recordamos lleva preinstalado el soporte para TCP/IP), existe un fichero WINSOCK.PAS, que redefine en código Delphi todas las funciones de la API Winsock. Este fichero, se encuentra en Delphi 2.0 y en Delphi 3.0 en el directorio: \Source\RTL\WIN

```
C:\DELPHI 2.0\Source\RTL\WIN
C:\DELPHI 3.0\Source\RTL\WIN
```

Creación de Winsock.pas

Si todavía trabajamos con Delphi 1.0 o simplemente queremos profundizar en la programación con *sockets*, será útil crear un objeto *socket* en Delphi. Para lograrlo, como hemos visto, lo más útil es definir una *unit* en Delphi (winsock.pas) que haga de interfaz entre la API Winsock y nuestros programas de Delphi.

Tenemos que decidir las funciones de la API del Winsock que vamos a *encapsular* en dicho objeto, que generalmente serán todas. Una vez definidas, habrá que llamarlas haciendo referencia a la DLL en que se encuentran. Lo vemos con un ejemplo: con la función *WSAStartup*. Después de leer la especificación de los *Windows Sockets*, comprobamos que hay que llamar a esta función para iniciar los servicios de la interfaz Winsock, antes de solicitar cualquiera de ellos. Los parámetros que requiere la función *WSAStartup*, se muestran a continuación:

```
int PASCAL FAR WSAStartup (WORD
      wVersionRequested, LPWSADATA
      lpWSAData );
```

donde,

wVersionRequested: es la versión superior de la API Windows Sockets que el programa puede utilizar (utilizada para negociar la versión más alta que ambos extremos de la conexión conocen).

lpWSAData: es un puntero a la estructura de datos *WSAData* que es la encargada de recibir los detalles de implementación de los *Windows Sockets*.

La estructura de datos *WSAData* es la siguiente:

```
struct WSAData {
  WORD  wVersion;
  WORD  wHighVersion;
  char
      szDescription[WSADESCRIPTION_LEN+1
      ];
  char
      szSystemStatus[WSASYSSTATUS_LEN+1]
      ;
  unsigned short  iMaxSockets;
  unsigned short  iMaxUdpDg;
  char FAR *  lpVendorInfo;
};
Por último, las constantes
      WSADESCRIPTION_LEN y WSASYSSTA-
      TUS_LEN se definen:
#define WSADESCRIPTION_LEN  256
#define WSASYSSTATUS_LEN  128
```

Nuestro trabajo consistirá en definir una nueva función, *WSAStartup* con parámetros de *Object Pascal* y que llame a la función correspondiente de la libre-

ría WINSOCK.DLL. Por tanto debemos definir en la parte *implementation* de la *unit*:

```
implementation

 function WSAStartup(Version: word; var
       LPWSAData: WSADATA) : integer; far;
       external 'WINSOCK';
```

Las palabras reservadas *far* y *external* son necesarias cuando realizamos una llamada a una función residente en una DLL. A continuación debe ir el nombre de la librería DLL sin la extensión: *'WINSOCK'*. El resto de los parámetros son idénticos a los de la función en C.

Será necesario también redefinir la estructura de datos WSAData, así como las constantes WSADESCRIPTION_LEN y WSASYSSTATUS_LEN:

```
type
LPWSAData = ^WSAData
WSAData = record
 Version: word;
 HighVersion: word;
 Description: array[0..WSADESCRIPTION_LEN]
       of char;
 SystemStatus: array[0..WSASYS_STATUS_LEN]
       of char;
 MaxSockets: u_short;
 MaxUdpDg: u_short;
 VendorInfo: PChar;
end;

const
 WSADESCRIPTION_LEN = 256;
 WSASYS_STATUS_LEN = 128;
```

Habrá que proceder igual con todas las funciones que quisiésemos tener implementadas en nuestro objeto Winsock. Lo más fácil es leer el fichero de cabecera en C (winsock.h) e ir transformando todos los tipos, constantes, variables y funciones a sus correspondientes en Pascal.

En el listado 1 (que está en el CD-ROM) aparece el aspecto que tendrá la *unit* "winsock.pas" en código Delphi, frente al que tiene el fichero "winsock.h", que se muestra en el listado 2.

Creación de un objeto "socket"

Con la *unit* "winsock.pas" terminada, crearemos un objeto *TSocket* que hará uso de las funciones redefinidas en dicha *unit* y creará una interfaz que envuelva dichas funciones. Las aplicaciones de red (es decir, los servicios de red) llamarán a la interfaz de este objeto *TSocket* olvidándose de los detalles de implementación del mismo.

Como hemos dicho con anterioridad, la primera tarea será iniciar los servicios de *Winsock* antes de utilizarlos. La función que realiza esto es *WSAStartup*. Definiremos una función "IniciarSocket" que llame a *WSAStartup*, le pase los parámetros correctos y devuelva *True* si ha tenido éxito la inicialización (*WSAStartup* retorna un valor cero si no hubo problemas y un código de error, que por simplicidad no analizaremos, en caso contrario).

El primer parámetro de *WSAStartup es Version* (utilizando el código en Pascal), que es la versión superior de la API *Windows Sockets* que el programa puede utilizar. Utilizaremos la 1.1 (por lo que el parámetro valdrá: $0101). El segundo parámetro, *LPWSAData,* será un puntero a la estructura de datos *WSAData*. Lo definimos en la parte *private* de la clase. Si la inicialización del *socket* se ha realizado con éxito, *WSAStartup* rellena la estructura de datos *WSAData* a la que apunta el puntero.

Para finalizar, hay que cerrar los servicios que se hayan abierto

Por último habrá que cerrar los servicios abiertos. Dos funciones son necesarias en este caso. La primera: *WSACancelBlockingCall*, cancela cualquier llamada bloqueante que no haya sido atendida (y devuelve cero si ha habido éxito); la segunda: *WSACleanup,* es la realmente encargada de clausurar los servicios *Winsock* y además efectúa las labores de limpieza de memoria (también devuelve cero si ha habido éxito). Como estas dos funciones son necesarias siempre, las llamaremos desde una función: *TerminarSocket,* que devolverá *True* si tuvo éxito. Al igual que con *IniciarSocket* y para simplificar no comprobaremos el error, si se produce alguno.

La clase *TSocket* quedará inicialmente como sigue:

```
unit Sock_sp;
interface
uses
 SysUtils, WinTypes, WinProcs, Messages,
       Classes, Graphics, Controls, Forms, Dialogs,
       Winsock;

TSocket_sp = class (TObject)
private
 PWSAData: LPWSAData;
public
 function IniciarSocket: boolean;
 procedure TerminarSocket;
end; {class TSocket}

implementation

function TSocket_sp.IniciarSocket: boolean
var error: integer;
begin
 if (WSAStartup($0101, PWSAData) = 0)
  Result := True
 else
  Result := False;
end;

function TSocket_sp.TerminarSocket: boolean
begin
 if WSACancelBlockingCall = 0 then
  Result := True
 else
  Result := False;
 if (WSACleanup <> 0) then
  Result := False;
end;

end.
```

Con estas dos funciones tenemos lo básico para iniciar y terminar un *socket*. Una vez iniciado debemos crear el *socket*

y después de usarlo debemos cerrarlo. Para crear un *socket*, existe la función *socket()*, que traducida a Pascal tiene la siguiente forma:

```
function socket (af, struct, protocol : integer) :
        TSocket;
```

El parámetro "af" informa del protocolo de red que se va a usar. En nuestro caso es TCP/IP por lo que utilizaremos la constante: PF_INET. Si el protocolo fuese otro, por ejemplo OSI (*Open Systems Interconnection*), usaríamos la constante predeterminada PF_OSI (ver archivo "winsock.h" o "wsock11.doc" en el CD-ROM de la revista).

Los cuatro parámetros de la función socket especifican todos los datos necesarios para establecer la conexión

El siguiente parámetro indica el tipo de *socket*. Básicamente hay dos tipos útiles: uno orientado a conexión (*stream socket*) y otro no orientado a conexión (*datagram socket*).

El primero es el utilizado por el protocolo TCP y será habitualmente el empleado, mientras el segundo sirve para comunicaciones en las que no se mantiene una conexión con el otro extremo, y por tanto los paquetes se envían uno independientemente del otro sin garantizar el orden de llegada.

La constante utilizada (ver "winsock.h") para indicar que queremos utilizar el *socket* TCP es: SOCK_STREAM.

El último parámetro ("protocol") define el tipo de protocolo. Fijándolo en cero no se especifica ningún protocolo en particular, y será por tanto el que usemos.

La función *socket()* devuelve el valor del *socket* creado.si ha tenido éxito y la constante INVALID_SOCKET en caso contrario.

Si hubo error podremos adivinar cual fue llamando a la función *WSAGetLastError*, que devuelve el código del último error producido. La función *CrearSocket* queda, por lo tanto, de la siguiente manera:

```
function TSocket_sp.CrearSocket: TSocket;
begin
  Result := socket(PF_INET, SOCK_STREAM,
        0);
  if (Result = INVALID_SOCKET) then
   ShowMessage('Error: ' +
        IntToStr(WSAGetLastError));
end;
```

Una vez utilizado el *socket* debemos cerrarlo al final del proceso. La función *closesocket()* desempeña esta labor:

```
function closesocket (s : TSocket) : integer;
```

El único parámetro que hay que pasarle es el valor del *socket* que queremos cerrar. El valor retornado será cero si el *socket* se cerró y SOCKET_ERROR en caso contrario.

Nuestra función *CerrarSocket* es pues muy sencilla:

```
function TSocket_sp.CerrarSocket(ValorDelSocket:
        TSocket): boolean;
begin
  if closesocket(ValorDelSocket) = 0 then
    Result := True
  else
   Result := False;
end;
```

Y aquí finalizamos esta introducción a la creación de nuestro objeto *socket*. En el próximo artículo terminaremos este objeto y haremos un pequeño programa de prueba del mismo.

Cabe aquí reseñar, antes de terminar, que los componentes *ClientSocket* y *ServerSocket* de Delphi 3.0 nos ahorran todo este trabajo, pues implementan el objeto *Socket* del lado cliente y el objeto *Socket* servidor.

Conclusión

En este artículo hemos estudiado qué son los *Windows Sockets* y por qué son necesarios cuando programamos aplicaciones para Internet.

Se ha dado una pequeña introducción a la programación de nuestro propio objeto Winsock, cuya función última es la de *encapsular* las funciones de la librería WINSOCK.DLL que nos hacen falta.

Hemos visto, además, las características y facilidades de programación para el Web que ofrece Delphi 3.0 en sus distintas versiones.

Después de examinar un poco la programación con *sockets*, que sólo supone el primer paso del proceso de creación de una aplicación para Internet, podemos apreciar mejor las herramientas y componentes que se regalan con Delphi 3.0 y que trataremos con más profundidad próximamente.

Cómo contactar con el Autor

Si alguien desea enviar algún comentario sobre este o sobre los anteriores artículos que se han publicado en Sólo Programadores, puede hacerlo enviando un mensaje de correo electrónico a la dirección de la revista:

solop@towercom.es

o a la dirección del autor:

miluen@redestb.es

enviando como "Subject" del e-mail: "DELPHI SOLOP".

Las Programación de objetos en Visual

Jordi Agost

Introducción

Visual Basic ha ido entrando poco a poco en el mundo de la programación orientada a objetos, haciendo siempre énfasis sobre la transición desde la programación por procedimientos. Y más en la versión 5.0 donde a las clases que aparecieron en la versión 4.0 de Visual Basic, se le ha añadido ahora el Polimorfismo.

Vamos a ver primeramente un poco la filosofía y términos de la programación orientada a objetos.

Desde un punto de vista general podríamos decir que una clase es un algo tal que describe un grupo de objetos similares. Por ejemplo, y moviéndonos dentro del marco empresarial podríamos decir que todos los empleados de una compañía dada son objetos de la clase empleados. Siguiendo con el mismo ejemplo también podríamos decir que todas las delegaciones de dicha compañía son objetos de la clase delegación. Así podemos decir que la Delegación de Madrid es una instancia de la clase "Delegaciones" o que el empleado "Pedro González" es una instancia de la clase "Empleados".

Siguiendo con el ejemplo podríamos afirmar, además, que cada clase definirá las propiedades y comportamientos de sus objetos. Por ejemplo la clase "Empleados" puede tener unas determinadas propiedades como podrían ser los datos personales, la categoría, el sueldo... y unos determinados comportamientos o acciones, por ejemplo el cálculo del sueldo.

En un lenguaje más propiamente informático diríamos que una clase define las propiedades (o atributos)y métodos (los comportamientos o acciones) de todos los objetos que han sido creados (o instanciados) de la clase.

Asimismo cualquier objeto instanciado o creado a partir de una clase le llamaremos instancia.

También podríamos decir que Visual Basic es un lenguaje de programación que desde siempre ha usado clases. Pensemos por un momento por ejemplo en las etiquetas o labels, en realidad cuando estamos añadiendo un label a un form determinado lo que estamos haciendo es creado una instancia de la clase label.

De qué clase es un objeto

A veces necesitaremos saber a que clase pertenece un determinado objeto para saber por ejemplo si le podemos aplicar una propiedad determinada o no. Para saberlo podemos utilizar las instruccio-

En este artículo veremos una introducción a la programación de objetos en VB, empezaremos primeramente con las características de los objetos mismos, luego veremos cómo construir clases y cómo tratar la herencia para ver finalmente cómo VB emplea el polimorfismo.

nes TypeOf y TypeName. TypeOf solo puede utilizarse en instrucciones del tipo If ... Then ... Else y debe incluir el nombre de la clase directamente en el código, como en el siguiente ejemplo:

```
If TypeOf Command3D1 Is CheckBox Then
...
```

En cambio la función TypeName la podemos utilizar en cualquier lugar del código y nos devolverá el nombre de clase como una cadena con lo que podremos compararla con una variable de tipo string.

Múltiples acciones con un objeto

Si lo que deseamos es hacer diferentes acciones sobre el mismo objeto, como por ejemplo establecer diferentes propiedades normalmento lo haríamos de la siguiente forma:

```
Private Sub Form_Load()
  Command1.Caption = "Prueba"
  Command1.Visible = True
  Command1.Enabled = True
End Sub
```

Pero una forma más práctica de obtener el mismo resultado sería:

```
Private Sub Form_Load()
  With Command1
        .Caption = "Prueba"
        .Visible = True
        .Enabled = True
  End With
End Sub
```

Otra característica propia de los objetos, es que muchos de estos tienen propiedades por defecto, con el fin de simplificar el código y hacerlo más legible. Por ejemplo, si tenemos un control de cuadro de texto llamado Text1 y escribimos lo siguiente:

```
Text1.Text = "Prueba"
```

Es lo mismo que escribir:

```
Text1 = "Prueba"
```

También podemos utilizar las propiedades por defecto si trabajamos con una variable de objeto, y ésta almacena una referencia a un objeto dado. Esto se especificaría con el siguiente código de programa:

```
Private Sub Command1_Click()

      Dim obj as Object
      Set obj = Text1
      Obj = "Prueba"

End Sub
```

Aunque hemos de ir con un cierto cuidado cuando utilicemos variables de tipo Variant para hacer lo anteriormente dicho. VB trata de forma especial las variables de tipo Variant porque estas pueden contener datos de muchos tipos diferentes. Por ejemplo podemos leer el valor de una propiedad predeterminada pero no establecerlo. Por ejemplo si dentro de un procedimiento declaramos lo siguiente:

```
Dim v As Variant
```

Y establecemos la propiedad determinada de Text a "Prueba":

```
Text1 = "Prueba"
```

Asignamos la variable y la mostramos en pantalla:

```
Set v = Text1
MsgBox v
```

Pero si hacemos:

```
v = "Prueba2"
MsgBox v
```

El último MsgBox que mostramos en pantalla nos dará como resultado la cadena "Prueba2", y Text1 no habrá cambiado.

Cómo crear clases

Ahora que tenemos una visión global sobre los objetos y como tratarlos vamos a ver como podemos crear las clases, para añadirlas a nuestros proyectos.

Para definir o crear una clase tenemos que seguir los siguientes pasos:

1. Insertamos un modulo de clase
2. Definimos sus propiedades
3. Creamos los métodos para la clase
4. Creamos los eventos.

Para el punto número 1, insertar un módulo de clase, haremos un nuevo proyecto en VB y luego con el menú Insertar, insertamos un Módulo de clase. Una vez hecho esto, editamos las propiedades de la clase (menú Ver y Propiedades o F4) y ponemos el nombre que nos interese a la clase, para nuestro ejemplo usaremos Cprueba.

Definiendo las propiedades

- **Datos miembro.**

Un módulo de clase, asimismo, podrá como cualquier otro módulo tener variables o elementos de datos asociados (tal y como un módulo tendría sus variables a nivel de módulo). A cada variable o elemento le llamaremos datos miembros. Algunos de estos datos serán para guardar las propiedades de la clase y se podrán acceder desde cualquier punto del programa y otros datos miembro serán invisibles para otras partes de la aplicación ya que sólo los usará la clase dada. Lograremos este último punto declarando a dichos datos privados, con la palabra reservada Private.

Además cabe decir que cada objeto creado o instanciado desde una clase tendrá una copia separada de sus datos miembros. Esto quiere decir que si cambiamos el valor de un dato de un objeto,

dicho cambio no repercutirá en los otros objetos del programa.

- **Propiedades de una clase.**

Cuando deseemos que una determinada propiedad de una clase sea visible a toda la aplicación lo lograremos con los procedimientos de propiedades. Dichos procedimientos son iguales que cualquier procedimiento de tipo Subrutina o Función pero con la característica de que están diseñados expresamente para la función de enseñar las propiedades al resto de la aplicación. Existen principalmente tres tipos de procedimientos de propiedades: Procedimientos de propiedades Get: dicho procedimiento nos servirá siempre que queramos que la aplicación pueda leer los valores de una propiedad. Como nota cabe decir que también podríamos incluir en dicho procedimiento los cálculos que quisiéramos para devolver un valor concreto.

Procedimientos de propiedades Let: este procedimiento permitirá a otras partes de la aplicación establecer un valor de una propiedad de la clase. En dicho procedimiento normalmente se suele almacenar el código responsable de la validación o conformación de datos.

Procedimientos de propiedades Set: constituyen un caso especial y particular del procedimiento de propiedades Let y se usan cuando la propiedad de la cual establecemos un valor, hace referencia a un objeto determinado.

Cabe destacar que podemos hacer que una propiedad sea solo de lectura o solo de escritura modificando sus procedimientos de propiedades Get, Let o Set. Vamos a ver un ejemplo de como implementar las propiedades. Imaginemos que queremos una clase que nos proporcione información de los empleados de una empresa. La información que deseamos guardar es por una parte el nombre del empleado, su sueldo y su categoría. Entonces vamos a crear primeramente los datos miembros de la clase correspondientes a la información que deseamos guardar:

```
Option Explicit

Private m_Nombre As String
Private m_Sueldo As Integer
Private m_Categoria As Integer
```

Además, cabe destacar que dichos datos miembros sólo serán accesibles desde dentro de la clase, ya que son datos privados (tal y como indica la palabra reservada Private).

Una vez realizado dicho paso pasaremos a establecer los procedimientos de propiedades para cada propiedad o atributo de la clase, teniendo en cuenta que si queremos poner alguna condición en la adquisición de datos, tendremos que poner el código adecuado para cada ocasión (ver como ejemplo el procedimiento Let de la propiedad Categoria).

```
Public Property Get Categoria() As Integer
Categoria = m_Categoria
End Property

Public Property Let Categoria(Cat As Integer)
If Cat > 1 Or Cat < 10 Then
  m_Categoria = Cat
End If
End Property

Public Property Get Nombre() As String
Nombre = m_Nombre
End Property

Public Property Let Nombre(Nombre As String)
m_Nombre = Nombre
End Property

Public Property Let Nombre(Nombre As String)
m_Nombre = Nombre
End Property

Public Property Get Sueldo() As Integer
Sueldo = m_Sueldo
End Property

Public Property Let Sueldo(Sueld As Integer)

'prevenimos que el sueldo sea >0
If Sueld > 0 Then
  mSueldo = Sueld
End If
End Property
```

Puede ser que al principio parezca un poco confuso, que parezca mejor el utilizar variables globales o una estructura de datos, pero debemos tener en cuenta que utilizando los procedimientos de propiedades tenemos las ventajas de que el código para la validación de datos o formato de los mismos puede ser encapsulado dentro de los procedimientos de propiedades, además podemos fácilmente establecer propiedades de solo lectura o solo escritura (lo hacemos sin asociar el procedimiento Get o el Let). Otra ventaja seria el hecho de que podemos modificar los procedimientos de propiedades sin tener que modificar los códigos que utilizan dichas propiedades.

Creando los métodos

Los métodos de una clase definirán el comportamiento de todos los objetos creados de una clase dada. La aplicación no verá la implementación del método, aunque su funcionalidad estará visible por toda la aplicación. Es como si por ejemplo usásemos en el objeto Label comentado anteriormente el método Move, nosotros no sabemos como se ha implementado dicho método, aunque si sabemos la funcionalidad que tiene para nosotros.

Para crear un método tenemos primero que nada determinar si éste va a ser público (se podrá acceder desde cualquier parte de la aplicación) o privado (sólo desde dentro de la clase), en segundo lugar tendremos que saber si retornará un valor o no para definirlo como una subrutina o como una función y por último lo escribimos.

Por ejemplo:

```
Public Sub Cambio_Categoria(Incremento As
      Integer)
 Categoria = Categoria + Incremento
End Sub
```

Creando los eventos

Los módulos de clases, tienen dos eventos: Initialize y Terminate. Dichos eventos nos aparecerán automáticamente cuando creemos una clase.

El evento Initialize se usa para inicializar los objetos creados desde la clase. Cuando un cierto se crea, lo primero que hará VB será ejecutar el código de dicho evento, incluso antes de establecer cualquier prioridad o de ejecutar cualquier método. Dicho evento sería similar a lo que en otros lenguajes de orientación a objetos sería el constructor de la clase.

Cuando todas las referencias a un objeto han sido canceladas entonces se ejecuta el evento Terminate.

Por ejemplo si queremos asignar una categoría inicial haríamos:

```
Private Sub Class_Initialize()
  m_Categoria = 1
End Sub
```

Utilizando la clase

Hasta ahora hemos visto como crear clases, aunque la clase por si sola no hará nada. Nosotros trabajamos siempre con instancias de la clase, por lo tanto necesitamos el código para crear instancias de una clase. Tenemos dos formas de crear una instancia de una clase:

```
Private m_Prueba as New Cprueba
```

O bien:

```
Private m_Prueba as Cprueba
Set m_Prueba = New CPrueba
```

En la primera forma declaramos una variable y le asignamos el objeto, y en la segunda forma la primera línea declarará la variable y la segunda asignará el objeto a la variable.

En nuestro ejemplo en particular:

```
Option Explicit

Private m_Prueba1 As CPrueba
Private m_Prueba2 As CPrueba
```

Para la declaración de variables, y luego en el procedimiento de carga del formulario les asignamos un valor:

```
Private Sub Form_Load()

 Set m_Prueba1 = New CPrueba
 Set m_Prueba2 = New CPrueba

End Sub
```

Para acceder a las propiedades de una clase el formato a emplear es el siguiente:

```
Objeto.Propiedad = Valor
```

En nuestro caso si suponemos que tenemos un botón para poner las propiedades, entonces su código sería:

```
Private Sub Command1_Click()

m_Prueba1.Nombre = TextNombre1.Text
m_Prueba1.Sueldo = Val(TextSueldo1.Text)
m_Prueba1.Categoria = Val(TextCategoria1.Text)

End Sub
```

Y por último nos quedaría terminar la referencia al objeto que hemos creado, para así liberar la memoria que hemos ocupado y los recursos del sistema. La sintaxis para realizar dicho paso sería:

```
Set Variable = Nothing
```

En nuestro ejemplo tenemos dicho código asignado a los botones "Destruir Clase 1" y "Destruir Clase 2", pero lo más lógico cuando estamos programando es que dicho código se encuentre por ejemplo en el procedimiento Form_Unload.

Guardando los datos de una clase en un fichero

Con lo aprendido hasta ahora nos encontramos con el problema de que en el momento que todas las referencias a un objeto han desaparecido, dicho objeto se destruye, y con él todos los datos que poseía.

Para evitar que suceda esto, nos veremos obligados a grabar los datos en un fichero o en una base de datos. La mejor forma para llevar a cabo dicho proceso es crear una clase genérica que haga la tarea de grabar los diferentes objetos o de leerlos según convenga.

Aunque si construimos una clase genérica tenemos el problema de que dicha clase no conocerá los elementos a ser leídos o grabados de cada objeto (y estos pueden ser muy diferentes y tener tratamientos diferentes, por ejemplo, strings, integers...), por lo que cada objeto tendrá que tener sus propios métodos para grabar sus datos.

Entonces podemos preguntarnos, ¿cuáles son las ventajas de tener una clase genérica? Principalmente podemos decir que la claridad y la abstracción a la hora de programar. Por ejemplo sólo tendremos un lugar en el código donde abriremos y cerraremos los ficheros. Sólo habrá un lugar donde chequearemos errores tales como si existe el fichero, si está siendo usado...

Además, podríamos cambiar en un determinado momento la forma en que se lee un fichero sin cambiar apenas código en la aplicación (sólo en el método del objeto afectado), ya que siempre que quisiéramos grabar o leer algo, lo haríamos mediante esta clase general.

Suponiendo que deseamos crear una clase general para grabar nuestros datos en un fichero, creamos una clase genérica que se llame Cfichero con las siguientes variables miembro:

```
Option Explicit

Private m_NombreFichero As String
Private m_NumFichero As Integer
```

Una propiedad necesaria será la del nombre del fichero, dicha propiedad ha de ser de lectura y escritura por lo que implementaremos los dos procedimientos de propiedades (Let y Get). Su implementación sería:

```
Public Property Let Nombre(vNewValue)
        m_NombreFichero = Nombre
End Property


Public Property Get Nombre()
        Nombre = m_NombreFichero
End Property
```

Además, necesitaremos un método que nos escriba el objeto en el fichero de datos que hayamos elegido

```
Public Sub EscribeObjeto(obj As Object)

Kill m_NombreFichero
m_NumFichero = FreeFile
Open m_NombreFichero For Binary As
        #m_NumFichero
Select Case Err.Number
  Case 0 ' no hay errores.
    obj.EscribeDatos
  Case Else
    'tratamiento de errores
End Select

Close m_NumFichero

End Sub


Public Sub LeeObjeto(obj As Object)

m_NumFichero = FreeFile
Open m_NombreFichero For Binary Access Read
        As #m_NumFichero
Select Case Err.Number
  Case 0 ' no hay errores
    Do Until EOF(m_NumFichero)
      obj.LeerDatos
    Loop
  Case Else
        ' tratamiento de los diversos errores
End Select
Close #m_NumFichero

End Sub
```

Definiendo colecciones

Una colección de objetos es un grupo ordenado de elementos que puede ser procesado como un conjunto.

Una colección puede ser referenciada desde cualquier sitio de la aplicación como cualquier otro objeto.

Para poder crear una colección necesitamos una clase (llamada clase de colección)

```
Option Explicit

Private m_ColPruebas As New Collection
```

Depuración de los módulos de clase

Cuando entramos con la depuración de módulos de clase nos encontraremos con que ésta es un poco diferente de los programas normales con los que estamos acostumbrados a trabajar. Esto sucede porque un error en un módulo de clase siempre actúa como un error controlado, que es lo mismo que decir que siempre habrá un procedimiento en la pila de llamadas que pueda controlar el error, ya sea éste el módulo de clase o el procedimiento que ha invocada la llamada a la propiedad. Es decir, si una determinada función o método de una clase produce un error, no se nos mostrará dónde realmente se ha producido éste sino desde el lugar donde se ha llamado dicha la mencionada propiedad.

Para evitar esta situación VB, ha añadido la opción de interceptar errores "Interrupción en módulos de clases", además de las existentes "Interrupción en errores no controlables" y "Modo de interrupción en todos". Dichas opciones están disponibles en el menú Herramientas, cuadro de dialogo opciones y ficha General.

Vamos a ver un ejemplo práctico:

Supongamos por ejemplo, que el módulo de Class1 dispone del siguiente código:

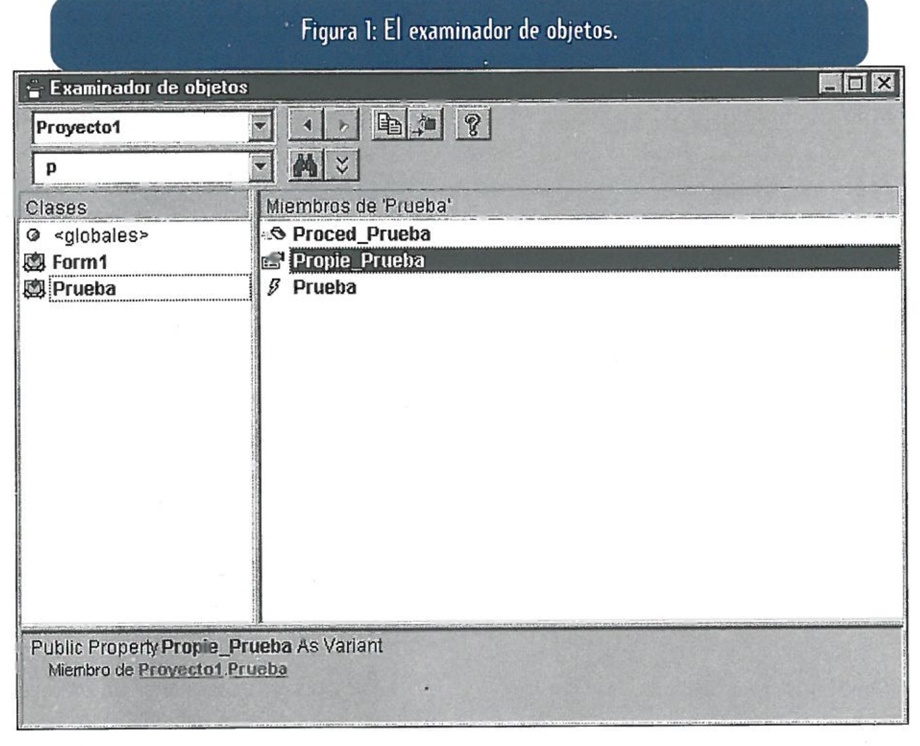
```
Public Sub Oops()
Dim intOops as integer
intOops = intOops / 0
End Sub
```

Por otra parte tenemos un módulo de clase o formulario estándar que invocará al miembro Oops:

```
Private Sub Command1_Click()
Dim c1 as New Class1
c1.Oops
End Sub
```

Si activamos la opción "Interrupción en errores no controlables" la ejecución no se detendrá en la división por cero. En dicha opción el error se producirá en el procedimiento que llama Command1_Click. Si utilizáramos la opción "Interrupción en todos los errores" se detendría en la división por cero, pero luego nos encontraríamos con que también se para en todos los errores, incluso en los que tenemos un código de control de errores, con lo que la depuración se dificultaría. Por eliminación obtenemos que la opción más lógica seria la de "Interrupción en módulos de clases", con dicha opción tenemos que el flujo del programa sólo se detendrá en los errores no controlados del módulo de clase , además si existe un controlador de errores la ejecución no se detendrá. Por otra parte es la opción predeterminada de VB. Y como nota final debemos añadir que si no existen módulos de clase dicha opción es equivalente a "Interrupción en errores no controlables".

Figura 1: El examinador de objetos.



El examinador de objetos

Para ver el examinador de objetos, en el menú ver elegimos examinador de objetos o bien presionamos F2, el examinador nos sirve para ver la información sobre las clases, métodos, propiedades...

La información nos es presentada en tres niveles, empezando por la parte superior podremos observar los proyectos y bibliotecas disponibles, incluidos nuestros propios proyectos.

Haciendo clic en una clase de la lista clases veremos su descripción en el panel inferior, las propiedades, métodos, eventos y constantes aparecerán en la lista miembros de la derecha.

Si hacemos clic en un miembro de la lista miembros veremos los argumentos y los valores de retorno del mismo. Otra forma de trabajar con el examinador a través del menú contextual, dicho menú nos presenta una alternativa a los botones copiar y ver definición del examinador de objetos, asimismo también permite abrir el cuadro de diálogo referencias y podemos ver además las propiedades del elemento que seleccionemos. También podremos establecer descripciones de nuestros propios objetos con Agregar descripciones de objetos.

Cuando la opción miembros del grupo está activada, todas las propiedades de un objeto aparecen agrupadas (también los métodos, eventos...) y cuando no está activada, la lista de miembros es de orden alfabético.

Y cuando la opción mostrar miembros ocultos las listas clases y miembros nos muestran la información marcada como oculta en la biblioteca de tipos. Se mostrarán en color gris claro.

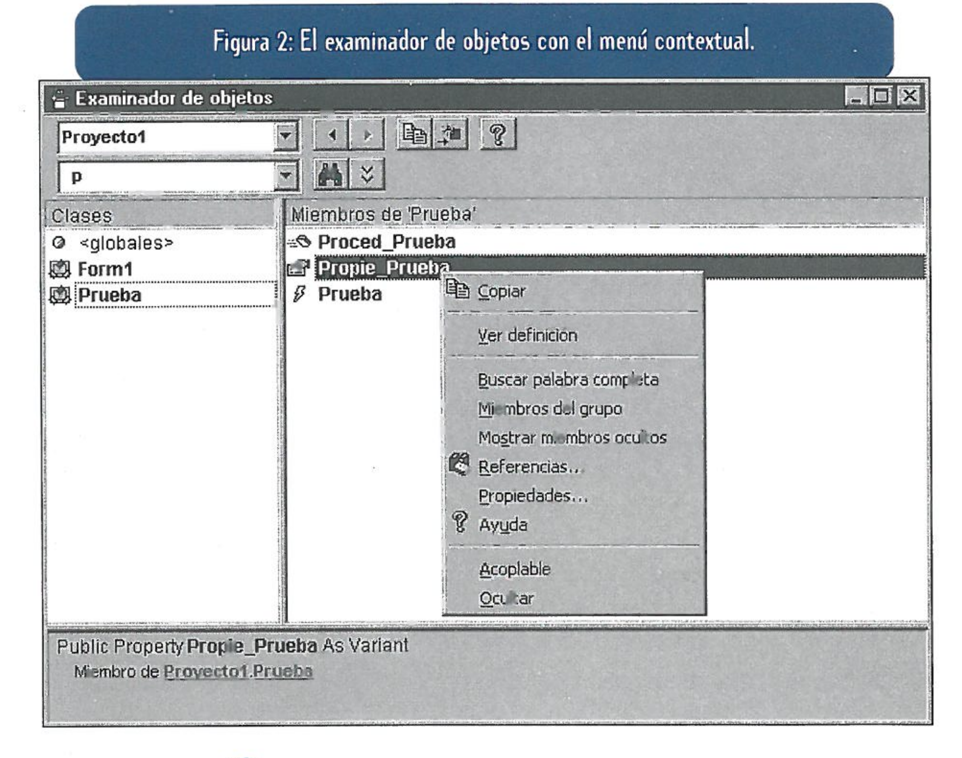
Polimorfismo

Vamos a tratar por último el polimorfismo dentro del ámbito de VB.

Polimorfismo dentro del ámbito del VB significa que muchas clases pueden proporcionar la misma propiedad o el mismo método y el que llama no tiene por qué saber la clase a la que pertenece el objeto antes de llamarla.

Por ejemplo si tenemos una clase que sea ingenieros y otra obreros las dos podrían tener un método "trabajar". Polimorfismo significa que podemos invocar el método "trabajar" sin saber si nos estamos refiriendo a un ingeniero o a un

Figura 2: El examinador de objetos con el menú contextual.



obrero (teniendo en cuenta que las dos clases tendrán trabajos diferentes). Logramos así una mayor claridad de código y una funcionalidad mayor ya que podemos "olvidarnos" del objeto al cual hacemos referencia.

La mayoría de lenguajes de programación utilizan el polimorfismo mediante la herencia, es decir, en nuestro caso las clase ingenieros y obreros, heredarían las características de una clase llamada trabajadores, cada clase (ingenieros y obreros) además haría valido su método trabajar, invalidando de esta forma el método trabajar de la clase trabajadores.

Visual Basic no utiliza la herencia para proporcionar polimorfismo, la proporciona mediante lo que llamaríamos múltiples interfaces ActiveX. Una interfaz es un conjunto de propiedades y métodos relacionados entre ellos. En VB creamos una interfaz llamada trabajador y la implementamos en la clases ingeniero y obrero. Una vez hecho este paso podremos invocar el método "trabajar" de cualquiera de las dos clases.

Implementación de una interfaz

Tal y como hemos dicho una interfaz será simplemente un conjunto de propiedades y métodos, ahora vamos a crear una interfaz llamado trabajadores y lo implementaremos en dos clases: Ingenieros y obreros.

Creamos la interfaz trabajadores si agregamos un módulo de clase al proyecto y le asignamos e insertamos el siguiente código:

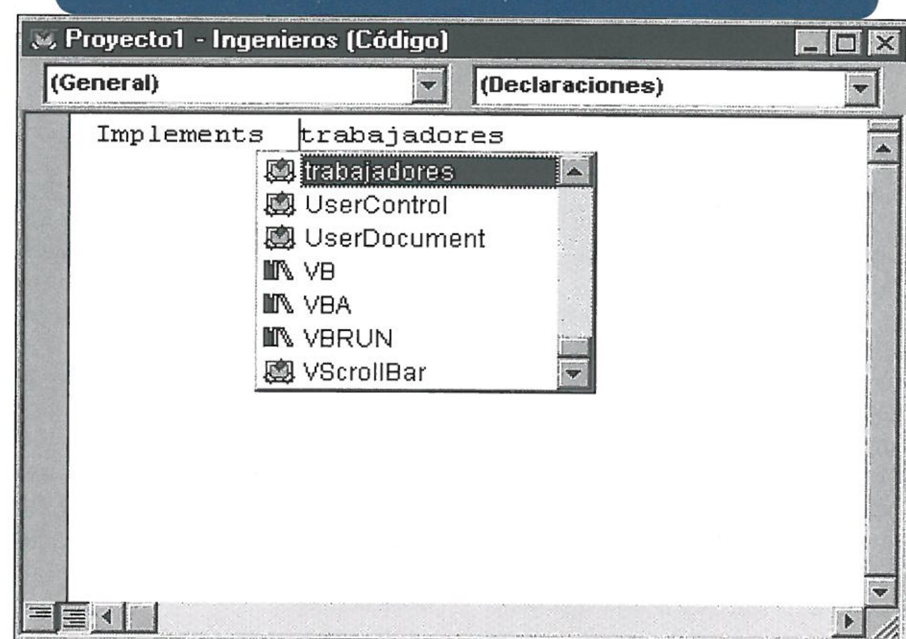
```
Public Sub Trabajar (ByVal horas as integer)

End Sub

Public Sub MoverDepartamento (ByVal Ob as
        Object)

End Sub
```

Figura 3: Clases y métodos de Trabajadores.



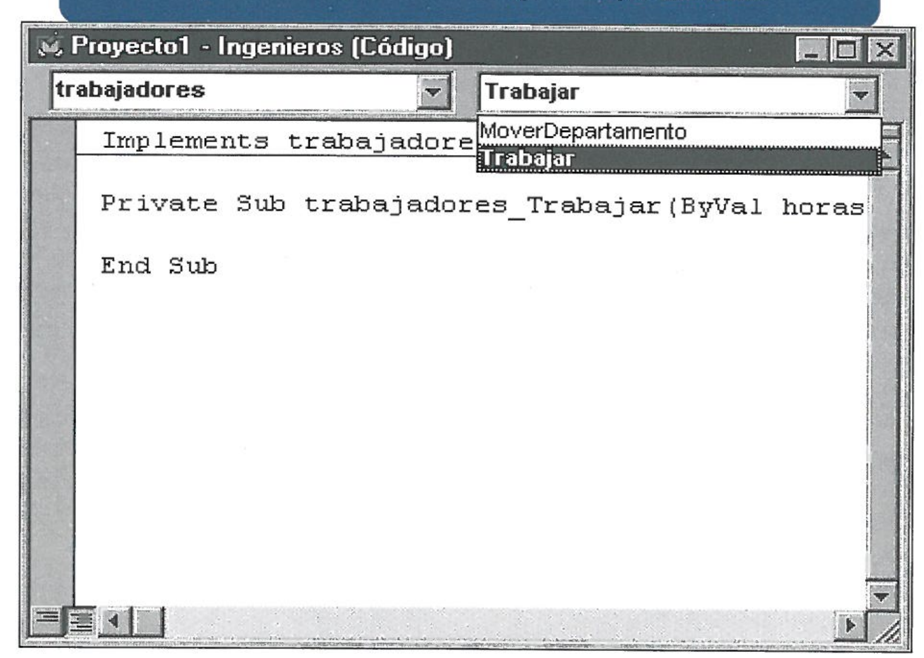
Destaquemos que dichos métodos no contienen código, trabajadores es una clase abstracta, que no contiene código de implementación. El objetivo de dicha clase es crear una plantilla de interfaz que añadiremos a otras clases.

Acto seguido vamos a agregar dos módulos de clase más, un primer módulo llamado ingenieros y otro obreros, y con los que implementaremos la interfaz trabajadores. Hacemos dicho paso con la instrucción Implements al principio del módulo:

```
Option Explicit

Implements trabajadores
```

Figura 4: Vemos como la clase ha adquirido los procedimientos.



Una vez realizado dicho paso, veremos como nos aparece una lista desplegable, de los objetos y luego podemos hacer clic para seleccionar "trabajadores de la lista"

Acto seguido si miramos los objetos en la ventana de código, en la lista desplegable de la derecha nos aparecerá los procedimientos de la interfaz trabajadores.

Como nota importante cabe destacar que al implementar una interfaz, la clase tiene que responder a cualquier propiedad o método de la interfaz, por lo tanto sacamos la conclusión de que una clase debe implementar todas las propiedades y métodos de una interfaz.

Ahora podríamos incluir todo el código que deseamos a la clase ingenieros

```
Private Sub
    trabajadores_MoverDepartamento(ByVal Ob
    As Object)
Debug.Print "Ingeniero moviendo departamento"
End Sub

Private Sub trabajadores_Trabajar(ByVal horas As
    Integer)
Debug.Print "Ingeniero trabajando"
End Sub
```

Ahora, en este punto la clase ingenieros posee dos interfaces: la interfaz trabajadores que hemos implementado, la cual posee dos miembros y la interfaz ingenieros predeterminada, que ahora por ahora no posee miembros.

Para ver un ejemplo más claro de cómo implementar estos métodos dentro de un programa cualquiera, imaginemos que tenemos un formulario y en el método Form_Load le añadimos dicho código:

```
Private Sub Form_Load()
Dim ing As Ingenieros
Dim obr As obreros
Dim tra As trabajadores
Set ing = New Ingenieros
Set obr = New obreros
Set tra = ing
Call tra.MoverDepartamento(obr)
Set tra = obr
Call tra.MoverDepartamento(obr)
End Sub
```

Como vemos en primer lugar declaramos tres variables, una de tipo ingenieros, una de tipo obreros y una de tipo trabajadores. Asignamos las variables de tipo ingenieros y obreros a dos objetos nuevos. Acto seguido la variable tra (de tipo trabajadores) y la cual puede tener cualquier referencia a cualquier objeto que implemente la interfaz trabajadores, la asignamos a la variable ingenieros primeramente para así llamar el método de MoverDepartamento.

Si ahora en la clase trabajadores queremos incluir una propiedad que sea edad, lo podemos hacer mediante una variable de tipo Public en la sección de declaraciones:

```
Option Explicit
Public Edad as Integer
```

En este momento las listas desplegables de procedimiento de los módulos de las clases ingenieros y obreros tienen los procedimientos de propiedad para implementar la propiedad Edad. Debemos decir que el empleo de este método (la declaración de una propiedad mediante una variable Public) está hecho simplemente para la comodidad del programador, internamente VB implementa las propiedades como parejas de procedimientos de tipo propiedad.

Si quisiéramos por ejemplo, poner un límite de edades, podríamos poner dicho código en el procedimiento Let de trabajadores_Edad, la única cosa que tenemos que tener en cuenta es que la interfaz coincida con la descripción de la biblioteca de tipos (es decir que coincidan los tipos de las llamadas).

Acto seguido hablaremos sobre las diferentes interfaces que puede tener un objeto. Para dicho supuesto vamos ahora a examinar el siguiente código:

```
Private Sub Command1_Click()
Dim ing as ingeniero
Dim tra as trabajadores
Set ing = New ingenieros
Set tra = ing
MsgBox TypeName(tra)
End Sub
```

Contrariamente a lo que pueda parecer la instrucción MsgBox nos presentará el mensaje "trabajadores".

De dicho ejemplo vamos a aprender algunas cosas. En VB cada clase tiene una interfaz predeterminada con el mismo nombre que el de la clase. Por ejemplo la clase Ingenieros tiene una interfaz por defecto llamada _Ingenieros (el subrayado indica que la interfaz se encuentra oculta en la biblioteca de tipos). Pero como dicha clase también implementa la interfaz trabajadores, por lo que decimos que tiene una segunda interfaz. No obstante, al final, debajo de cada interfaz el objeto continua siendo trabajadores, que es lo que nos viene a demostrar el ejemplo visto.

Cuando asignamos un objeto Ingenieros a una variable de tipo trabajadores, entonces VB preguntará al objeto Ingenieros si acepta la interfaz de trabajadores (a través del método llamado QueryInterface o QI), si la respuesta es si, el objeto se asignará a la variable, y a través de dicha variable sólo podremos acceder a los métodos y propiedades de la interfaz animal, en cambio, si la respuesta es negativa se producirá un error.

Pensemos ahora que sucede si asignamos una referencia de objeto a una variable genérica de objeto:

```
Private Sub Command1_Click()
Dim ing as Ingenieros
Dim tra as Trabajadores
Dim obj as Object
Set ing = New Ingenieros
Set tra = ing
Set obj = tra
MsgBox TypeName(obj)
End Sub
```

El resultado será Ingenieros. Así que hemos de tener en cuenta que la interfaz a la que la variable tendrá acceso será la última interfaz asignada.

Pert

José Antonio Mendoza y Francisco Gayá

En el artículo de este mes vamos a presentar al Pert, comentando su funcionamiento, sus usos principales, como se desarrolló y con que intención. Así mismo comentaremos como funciona el programa, que lo resuelve.

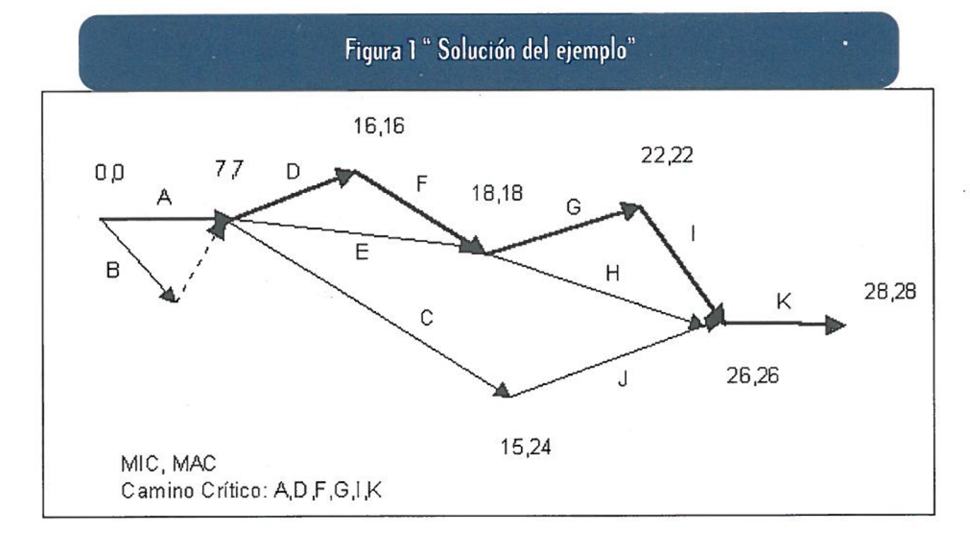
El Pert es un método de planificación de proyectos de amplia difusión, y aplicación práctica. El término PERT, viene de las iniciales de las palabras Inglesas Programme Evaluation and Review Technique. En mil novecientos cincuenta y ocho, en plena guerra fría la marina de los Estados Unidos de América se encuentra ante el desafío de construir submarinos nucleares, a los cuales se les dotará la capacidad de lanzamiento de los misiles Polaris. Este no es precisamente un objetivo sencillo, resulta necesario sincronizar y sumar los esfuerzos de muchas empresas, recursos, personas prever los costes y duración de las tareas, para lograr el objetivo marcado a tiempo y con los costes previstos, es decir sin gastar mas de lo presupuestado. Para resolver todo esto se desarrolló esta técnica.

El Pert aplica la máxima del divide y vencerás, un proyecto ambicioso, de gran presupuesto, de larga duración en el tiempo, de complejidad en su ejecución, e incertidumbre en los plazos, se divide en otros mas pequeños y así sucesivamente hasta que hemos descompuesto nuestro gran proyecto en una serie de pequeñas y abordables tareas.

Por lo tanto la unidad básica de un pert será la tarea, y esta vendrá fundamentalmente definida por su duración. Cuando un productor se decide a financiar una película lo que mas le preocupa son las semanas de rodaje, cuanto mas dure mas cara le resultará la película. En resumen en cualquier proceso productivo, el tiempo es dinero, y se tiende a acortar lo máximo posible el periodo de realización de cualquier proyecto.

Para lo cual y mediante el pert se intentará armonizar las tareas del proyecto para que se realicen el mayor número de ellas simultáneamente. Claro está habrá tareas que para poder empezar tendrán que esperar a que otras hayan terminado. Es el momento oportuno de contar el clásico ejemplo del ejecutivo agresivo que se levanta por la mañana para tomar un avión, y desde que se levanta hasta que embarca no para de encargar tareas. Veámoslo mas de cerca. Nuestro protagonista lo primero que hace al levantarse es llamar por teléfono a su secretaria para que le prepare la documentación que necesita para el viaje de negocios así como el billete de avión. Justo después avisa al mayordomo para que le prepare las maletas con la ropa necesaria para el viaje y al servicio para que le tenga listo el desayuno y el traje del día, en el mas breve espacio posible de tiempo. Justo después de dar todas esas ordenes nuestro protagonista va al cuarto de baño donde se afeita, se asea y se ducha, justo cuando termina va a su dormitorio donde encuentra listo el traje y se viste, se dirige posteriormente al comedor donde encuentra preparado un delicioso desayuno para empezar con fuerzas su dura jornada laboral. Al terminar de desayunar se encuentra con el equipaje listo, preparado por su mayordomo. En ese momento se

Figura 1 " Solución del ejemplo"



encamina hacia la oficina donde la secretaria le ha preparado los documentos y reservado el vuelo, lo recoge todo y se marcha directamente al aeropuerto.

La otra manera de realizar lo mismo es la secuencial, nuestro ejecutivo se levanta se ducha y encarga que le preparen el traje, cuando llega este se viste y al llegar al comedor encarga el desayuno, cuando termina de desayunar le encarga al mayordomo que le prepare el equipaje y así con todo. De lo que se deduce la gran cantidad de tiempo perdido por no haber planificado bien las actividades. De todo el ejemplo deducimos dos aspectos fundamentales, el primero la sincronización de las tareas, cuando el ejecutivo termine de vestirse tiene que tener el desayuno listo, y el otro tiene que ver con la necesidad de que las tareas no se prolonguen mucho, retrasando a todo el proceso.

Visto lo anterior adentrémonos en el método Pert. Este necesita saber el conjunto de tareas que componen un proyecto su duración y el orden de precedencia, es decir, que tareas deben haber terminado para que pueda empezar otras.

Para determinar la duración de una de las tareas suele emplearse una fórmula, que nos devuelve la duración más probable de la tarea, en función de la duración mas optimista, la mas pesimista y la media, la fórmula es la siguiente :

$$t_{duracion} = (t_{mejor} + 4^{*} t_{medio} + t_{peor}) / 6.$$

Una vez que nuestro proyecto lo tenemos dividido en tareas, que de estas conocemos una duración representativa de todas las posibles, y las precedencias de las mismas, podemos entonces comenzar la construcción del grafo.

" El Pert se empleó en el proyecto Apollo y en el de misiles Polaris"

Antes de ello, unas nociones necesarias de lo que es un grafo en el modelo pert y los conceptos que son necesarios para entender su funcionamiento. En primer lugar veremos el concepto de tarea, el cual se representará por una flecha de trazo continuo. El concepto de suceso, que consiste en la confluencia de dos o mas tareas en un instante de terminado. El concepto de restricción, es decir una tarea precede a otra o a un conjunto de ellas : A < D,E,F , la Tarea A precede en su ejecución a las tareas D, E, y F, o lo que es lo mismo, para que empiecen D,E, y F tiene que haber terminado A. Existen unas tareas virtuales de duración cero que para lo único que sirven es para indicar la precedencia de una tarea respecto de otra. Nos queda saber lo que es el camino crítico, el Mic y Mac.

El camino crítico es aquella sucesión de tareas que dan la duración mínima de ejecución de un proyecto, o lo que es lo mismo, en menos tiempo es imposible realizar el proyecto. El Mic de una tarea es el momento mas temprano de comienzo y el Mac es el momento mas tardío de cominezo de la misma. El Mic de un suceso es el máximo de los mics de las tareas que confluyen a él, el Mac de un suceso es el mínimo de los macs de las tareas que confluyen en él.

Una tarea se dice crítica cuando su mic y su mac coinciden. El camino crítico está formado pues por un conjunto de tareas críticas. Con todos estos conceptos ya podemos construir el grafo del proyecto, calcular los mics y macs de cada suceso y por lo tanto el camino crítico. Veámoslo con un ejemplo.

Tras el análisis de nuestro proyecto hemos llegado al resultado de que tenemos una serie de tareas con su duración, calculada según la formúla anateriormente expuesta y las restricciones de cada una de ellas.

A = 7 ; A < D, E, C
B = 6 ; B < C
C = 8 ; C < J
D = 9 ; D < F
E = 7 ; E < G, H
F = 2 ; F < G, H
G = 4 ; G < I
H = 3 ; H < K
I = 3 ; I < K
J = 1 ; J < K
K = 3

La duración viene expresada en una unidad de tiempo genérica, y el símbolo menor que indica que las tareas a la derecha pueden comenzar cuando termina la tarea de la izquierda. Lo primero que tenemos que hacer es construir el grafo. Para lo cual procederemos de la siguiente manera :

Cada tarea será representada por una flecha continua, y cuando tengamos que explictar una precedencia que no aparezca mediante las flechas continuas, lo haremos mediante una tarea virtual, representada por una flecha discontínua. Así tendremos un punto del que parten

dos flechas la de la tarea A y la de la tarea B. De A nacen tres tareas C, D, y E, pero B precede a C por lo que unimos la punta de B con la de A mediante una flecha discontinua. Continuando con el proceso de la punta de flecha de C nace una flecha la de J, de D nace la de F y de E nacen dos la de G y la de H, de G nacería la de I y desde la de H la de K, Tenemos puestas todas las de tareas reales, para indicar las precedencias que no hemos puesto nos queda poner flechas discontinuas desde la punta de J a la de I, desde la punta de F a la de E y desde la punta de I a la de H. Ahora eliminaríamos las tareas virtuales redundantes, aquellas que si las quitas no se montan dos flechas continuas.

Una vez que tenemos el grafo construido ponemos primero los mics y después los macs, los primeros de izquierda a derecha y los segundos de derecha a izquierda. Según las siguientes reglas :

1. Ponemos Mic = 0 para el suceso inicial, en nuestro caso tenemos dos sucesos iniciales A y B, pueden empezar sin que tenga que haber concluido ninguna tarea.

2. El mic de los sucesos inmediatamente posteriores al primero es igual a la duración de la tarea que los precede.

3. El Mic de las demás sucesos es el máximo de la suma de los mics anteriores mas la duración de la tarea actual, por los diferentes caminos que lleven a ella.

4. Una vez calculados todos los mic, sabiendo que en el suceso final el mic es igual al mac. En los sucesos anteriores en el timpo será el mínimo de la resta los macs posteriores en el tiempo y la duración de la tarea que los separa , teniendo en cuenta todos los caminos posibles.

Esto que parece muy complicado y difícil de entender a la primera lo veremos mas claro solucionando el ejemplo.

Lo primero A y B son dos tareas que empiezan a ejecutarse en el instante cero, como dijimos el mic del primer suceso es cero. Cuando termina de ejecutarse A pueden empezar a ejecutarse D, E y C y cuando termina de ejecutarse B Puede empezar a ejecutarse C, pero como B dura menos que A la que se impone es esta última y es la que marca el cominezo de las tres, el Mic del segundo suceso será 0+7 = 7.

Por lo tanto justo al terminar A empiezan a ejecutarse C, D y E. El tercer suceso será donde termina B y empieza C y a donde llega la tarea virtual que parte de la punta de A a la punta de B, el cual también tendrá un Mic de 7, ya que el máximo de 6 y 7 es siete. Así iríamos calculando los mics hasta llegar al del último suceso aquel donde termina K y entonces

empezaríamos a calcular los macs de cada uno de los sucesos. Como hemos dicho hace un momento el mic y el mac del ultimo suceso son el mismo, en nuestro caso 28 que es el instante en que termina la terea K. Para calcular el mac del suceso anterior a este mac le restamos la duración de la tarea K dando 28-2=26 t y así sucesivamente vemos mejor todo el proceso en el grafo de solución de nuestro proyecto.

Veamos ahora el programa en C que resuelve de manera automática todos los cálculos que hemos realizado anteriormente de manera manual :

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>

int main ( ) {
      /* Datos */

      float      duracion[26];              /* Duracion
      de cada tarea */
      char       dependencia[26][26];    /*
```

Cuando tenemos resuelto el problema, nos dedicamos entonces a optimizar nuestra asignación de precedencias y temporizaciones a las tareas que hemos diseñado. Lo que se pretende es armonizar lo mas posible las tareas y hacer mas pequeño el camino crítico, lo que se consigue asignando mas recursos a aquellas tareas que retrasan mas al proyecto. Existen un sin fin de modos de lograr dicho objetivo, uno de ellos es, que una tarea en vez de realizarla de forma secuencial se realice en paralelo, asignando dos líneas de producción a lo que antes tenía una sola. Lógicamente se tendrá que llegar a un compromiso para no gastar mas de lo que nos vamos a ahorrar por realizar el proyecto global en menor tiempo. Además de esta existen un gran número de maneras, fuera del objetivo de este artículo.

Veamos ahora una breve explicación del funcionamiento del software. El programa pregunta inicialmente por la dura-

Listado 1

```
Tabla de tareas precedentes a cada tarea */
      char      c_critico[26];
      unsigned n_tareas;                    /* Total de tareas */

      /* Resultados */
      float     MIC[26];
      float     MAC[26];
      float     duracion_minima_total;

      /* Variables auxiliares */
      unsigned          tarea_actual;
      unsigned          ix_preced;
      unsigned          tarea_precedida;
      int               flag_cambio;

      tarea_actual = 0;
      printf ("PERT\n\nDuracion (T) de las tareas\n");
      do {
            printf ( "T(%c)=" , 65 + tarea_actual );
            scanf( "%f" , &duracion[tarea_actual]);
            dependencia[tarea_actual][0] = 0 ;
      }
      while ( duracion[tarea_actual++]!= 0.0 );
      n_tareas = tarea_actual - 1;

      printf ("\nDefinicion de Restricciones\n");
      flushall();
      while ( 13 != (tarea_actual = toupper(getch())) ) {
            tarea_actual -= 65 ;
            if (tarea_actual < n_tareas) {
                  printf ( "%c < ", 65 + tarea_actual ) ;
                  strupr(gets(dependencia[tarea_actual]));
            }
      }
      /* Calculo del MIC */

      /* Establecer el valor por defecto */
      for ( tarea_actual = 0 ; tarea_actual < n_tareas ; tarea_actual++ ) {
            MIC[tarea_actual] = 0.0 ;
      }
      /* Bucle que se prosigue hasta la estabilizacion de los valores */
      do {
            flag_cambio = 0;

            /* Bucle de proceso de cada tarea */
            for (tarea_actual = 0; tarea_actual < n_tareas; tarea_actual++) {
                  if (strlen(dependencia[tarea_actual]))
                  /* Bucle de proceso de las restricciones por cada tarea */
for (ix_preced = 0 ; ix_preced < strlen(dependencia[tarea_actual]) ; ix_preced++) {
                        tarea_precedida = dependencia[tarea_actual][ix_preced]-65;
      if ((MIC[tarea_actual] + duracion[tarea_actual]) > MIC[tarea_precedida]) {
      MIC[tarea_precedida] = MIC[tarea_actual]+duracion[tarea_actual];
                                    flag_cambio = 1 ;
                              }
                  }
            }
```

Listado 1 (Continuación)

```
    } while (flag_cambio) ;

    /* Calculo de la duracion minima */
    duracion_minima_total = 0 ;
    for ( tarea_actual = 0 ; tarea_actual < n_tareas ; tarea_actual++ ) {
            if (duracion_minima_total < (MIC[tarea_actual] + duracion[tarea_actual]) )
                    duracion_minima_total = MIC[tarea_actual] + duracion[tarea_actual];
    }
    /* Calculo del MAC */

    /* Establecer el valor por defecto */
    for ( tarea_actual = 0 ; tarea_actual < n_tareas ; tarea_actual++ ) {
            MAC[tarea_actual] = duracion_minima_total - duracion[tarea_actual] ;
    }

    /* Bucle que se prosigue hasta la estabilizacion de los valores */
    do {
            flag_cambio = 0;

            /* Bucle de proceso de cada tarea */
            for (tarea_actual = 0; tarea_actual < n_tareas; tarea_actual++) {
                    if (strlen(dependencia[tarea_actual]))
                            /* Bucle de proceso de las restricciones por cada tarea */
                            for (ix_preced = 0 ; ix_preced < strlen(dependencia[tarea_actual]) ;
ix_preced++) {
                                    tarea_precedida = dependencia[tarea_actual][ix_preced]-65;
                                    if ((MAC[tarea_precedida]-duracion[tarea_actual]) <
MAC[tarea_actual]) {
                                            MAC[tarea_actual] = MAC[tarea_precedida]-dura-
cion[tarea_actual];
                                            flag_cambio = 1 ;
                                    }
                            }
            }
    } while (flag_cambio) ;

    /* Calculo del camino critico */
    ix_preced = 0;
    for ( tarea_actual = 0 ; tarea_actual < n_tareas; tarea_actual++ ) {
            if (MIC[tarea_actual]==MAC[tarea_actual])
                    c_critico[ix_preced++] = tarea_actual + 65;
    }
    c_critico[ix_preced] = 0;

    /* Presentacion de resultados */
    printf("\nResultados\n");
    for ( tarea_actual = 0 ; tarea_actual < n_tareas; tarea_actual++ ) {
            printf ("T(%c)=%6.2f    MIC=%6.2f    MAC=%6.2f\n", 65+tarea_actual,
                    duracion[tarea_actual], MIC[tarea_actual],
                    MAC[tarea_actual]);
    }
    printf ("Camino critico: %s  Duracion total:%6.2f",c_critico,duracion_minima_total);
    return (0);
}
```

ción de las tareas, para indicar que ya no quedan mas tareas que introducir, basta indicar una tarea de duración cero. A la hora de introducir las dependencias entre las tareas indicar la letra de la tarea que precede a las otras y posteriormente el programa pregunta por la tarea o tareas que la siguen, estas se introducen una detrás de otra sin espacios ni comas y en orden alfabético creciente, es decir ABFH.. Todas las reglas de tareas se introducen también por orden alfabético primero la de la A después la de la B así hasta que se terminen, para lo cual basta pulsar Enter en vez de una letra. Así mismo en los fuentes existe una versión en Basic de este programa, la cual supongo será útil a todos aquellos que dispongan de alguna calculadora que interprete dicho lenguaje. El basic del programa es cien por cien compatible con el de Gwbasic de Microsoft y el de las calculadoras Casio de la serie FX-8XXP. Solo añadir que ahora para terminar de introducir dependencias hay que teclear el cero en vez de Enter.

Conclusión

El método Pert está presente en la mayoría de los proyectos que tengan una mediana envergadura. La administración Americana exige que todo proyecto que se presente a cualquiera de los que oferta, adjunte un modelado de tareas según el pert. Para el control de plazos de ejecución, costes, recursos etc. Gracias a esta técnica se ahorra mucha cantidad de dinero, a la hora de su realización lo que redunda en precios mas baratos en los productos terminados, llegando a mas público y a mejor precio.

Bibliografía

- " Dirección de operaciones : Aspectos estratégicos en la producción" Autor : José A. Dominguez Manchueca.



Correo del lector

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Pregunta

1.- En el número monográfico de Internet que tengo en mis manos, solamente viene una aplicación para Linux, sólo 1 entre cerca de 50 aplicaciones. Yo creo que el porcentaje es bajísimo, y ya que todos sabemos que el sistema operativo Linux está mejor concebido para Internet que Windows 95/NT, la cosa va mal. Me gustaría que vinieran más aplicaciones de Linux en los CD's. En otras revistas vienen más cosas de Linux que en esta.

2.- Por cierto: el número especial que venía con la distribución Debian, que no pude comprar porque se agotó enseguida, parece ser que va ser reeditado de nuevo, por la publicidad que veo en la revista, que dice que va a estar muy pronto otra vez en el quiosco, pero ¿cuándo?. Hay muchos lectores que nos quedamos sin ese ejemplar, y nos gustaría poder conseguirlo.

3.- A ver si de una vez se deciden entregar el CD de regalo con una fundita, aunque sea una muy cutre de plástico, que por lo que vale la revista, ya se podían estirar un poco. No pido que la pongan más barata, aunque sería una ayuda para mi economía. Pero vamos, que la funda esa no valdrá más unas pocas pesetas.

Gracias por leer esto.

Respuesta

Pues bien, hemos seleccionado este mensaje de correo, ya que parece reflejar las inquietudes de muchos de vosotros. Hemos recibido algunos similares, referentes a alguno de los puntos que en este correo están todos agrupados. De modo que vayamos por partes en la contestación:

Por el punto 1: Si bien es cierto que Linux está mejor concebido para Internet que otros sistemas operativos (entre ellos los consabidos Windows 95 y NT de la omnipotente Microsoft), también es verdad que el parque efectivo de instalaciones de Linux es mucho menor que el de Windows. De modo que en la revista no hacemos más que un eco de lo que nos muestra el mercado día a día. En cualquier caso, y dado que parece existir una importante demanda de programas para Linux, esta acertada petición no va a caer en saco roto y haremos un esfuerzo por seleccionar más material para este estupendo sistema operativo.

En el CD de este mes ya hay algo más y seguiremos incrementando la presencia de aplicaciones Linux mes a mes. De hecho, si alguien más se adhiere a esta petición, puede enviar un correo a la revista. De este modo siempre estaremos informados de lo que necesitais y podemos responder adecuadamente.

Por el punto 2: Correcto. El número de Debian se agoto a unas velocidades insospechadas y nos cogió algo desprevenidos. De todas formas, en ello tienen algo de culpa los distribuidores, por no pedir más números a tiempo.

En nuestro almacén general, sin embargo, aún disponemos de números especiales, de modo que cualquier lector que lo desee, los puede pedir directamente a Tower Communications, ya sea mediante correo convencional (la dirección está al principio, en el Staff) o por correo electrónico a **suscrip@tower-com.es** o, finalmente, por teléfono al (91) 661 42 11 (Sta. Isabel). Le serán enviados a vuelta de correo o en la forma que lo pida y desee.

Por el punto 3: Pues si, tienes razón. Llevamos algún tiempo dándole vueltas al asunto del almacenaje de los CD-ROM de la revista, y al hecho que al final del año se han juntado los suficientes como para que las posibilidades de andar "desparramados" por todas partes en la habitación son relativamente altas.

Debido a ello, nos hemos decidido a fabricar un archivador anual en cartón, que estará disponible durante el mes de diciembre, para que todos los CD tengan su sitio y haya un sitio para cada CD (actualizando el proverbio).

Esperamos que esta solución sea de vuestro agrado.

CONTENIDO DEL CD

Xfree, X Appeal, 3D Winbench, clientes CICS

Otro mes más estamos juntos de nuevo preparados para hacer uso del CDROM de este número. En este os veréis otra vez abrumados por la cantidad de herramientas Java que se han incluido, pero es que las empresas de software están completamente volcadas en este lenguaje, y si ellas apuestan fuerte por la creación de Sun, nosotros también.

Sin embargo, no os penséis que esto es lo único que podréis encontrar en el interior, pues los forofos de Linux tienen casi medio CD dedicado a ellos. Pero vamos, dejémonos de explicaciones y pasemos a describir qué podéis encontrar en esta entrega.

Clientes CICS

En el apartado de redes se encuentran una serie de clientes para acceder a los distintos servicios de las plataformas CICS. CICS es un servidor del nivel de aplicación que proporciona procesado de transacciones de capacidad industrial, manejo de transacciones para aplicaciones que necesiten trabajar en entornos críticos de tiempo real, en plataformas tanto IBM como de otros fabricantes diversos. Es capaz de operar tanto en redes de area local como en WANs, manteniendo la integridad y la dependencia; es escalable de diez a 10,000 usuarios y puede ser administrado desde un solo punto, incluso en completas configuraciones.

CICS abre una puerta a la compatibilidad en plataformas como IBM, AIX, MVS, OS/2, OS/400 y VSE, así como entornos no IBM como Windows NT, HP, Digital y Sun.

Los clientes que proporcionamos en este CD-ROM, facilitan el acceso a estas redes y los suministramos para múltiples plataformas, como AIX, DOS, Macintosh, OS/2, Sinix, Solaris, Windows 3.x, 95 y NT

Linux: Xfree86 y X Appeal

XFree86 es una marca registrada del "XFree86 Project, Inc." una organización altruista que proporciona servidores X Window System así como algunas utilidades complementarias de soporte técnico para una gran cantidad de sistemas operativos en PCs y otras plataformas.

Los servidores X, programas clientes, documentación, etc. proporcionados por el proyecto XFree86 son en conjunto conocidos como XFree86; además todos ellos son distribuidos por diferentes vías (especialmente Internet) con su código fuente y están libres de cualquier cargo.

El proyecto XFree86 se financia absolutamente a base de donaciones, de hecho si estás interesado en contribuir puedes hacerlo en http://www.XFree86.org/donations.html o enviando correo electrónico a bod@xfree86.org. La última versión de las XFree86 es la 3.3.1, que proporcionamos en el presente CD-ROM. Está basada en la X11R6.3pl2 y posee fecha de Agosto de este año.

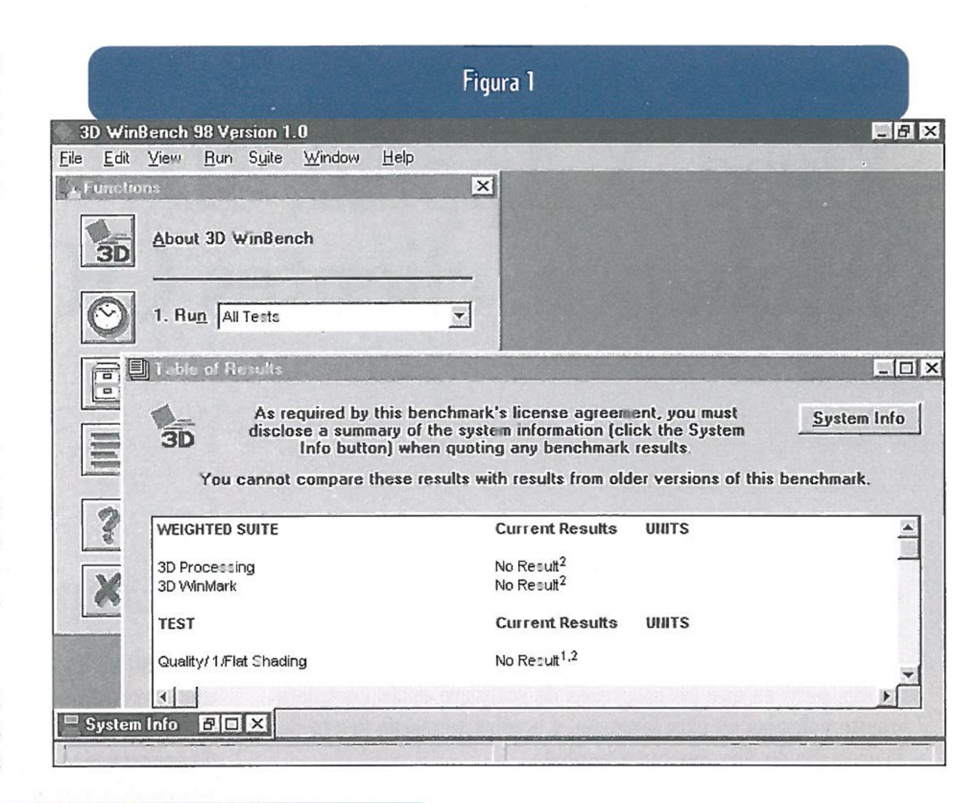
X Appeal es un servidor del sistema X Window ejecutable bajo MS-DOS. La actual versión de X Appeal está basada en los fuentes X11R6 con ampliaciones y corrección de errores procedentes de XFree86 3.1.

Java

En el apartado de programación, este mes incluimos una serie de aplicaciones de la conocida casa Sun Microsystems, todas ellas orientadas a la programación profesional bajo Java, tanto en el desarrollo de programas cliente/servidor como las que se encuentran dirigidas al testeo exhaustivo de prototipos para su utilización comercial.

Las instalaciones corresponden a las versiones de W95 y NT. Se pueden encontrar también estas utilidades para otros sistemas como Solaris, HP-UX, AIX, etc... en los directorios correspondientes.

Figura 1

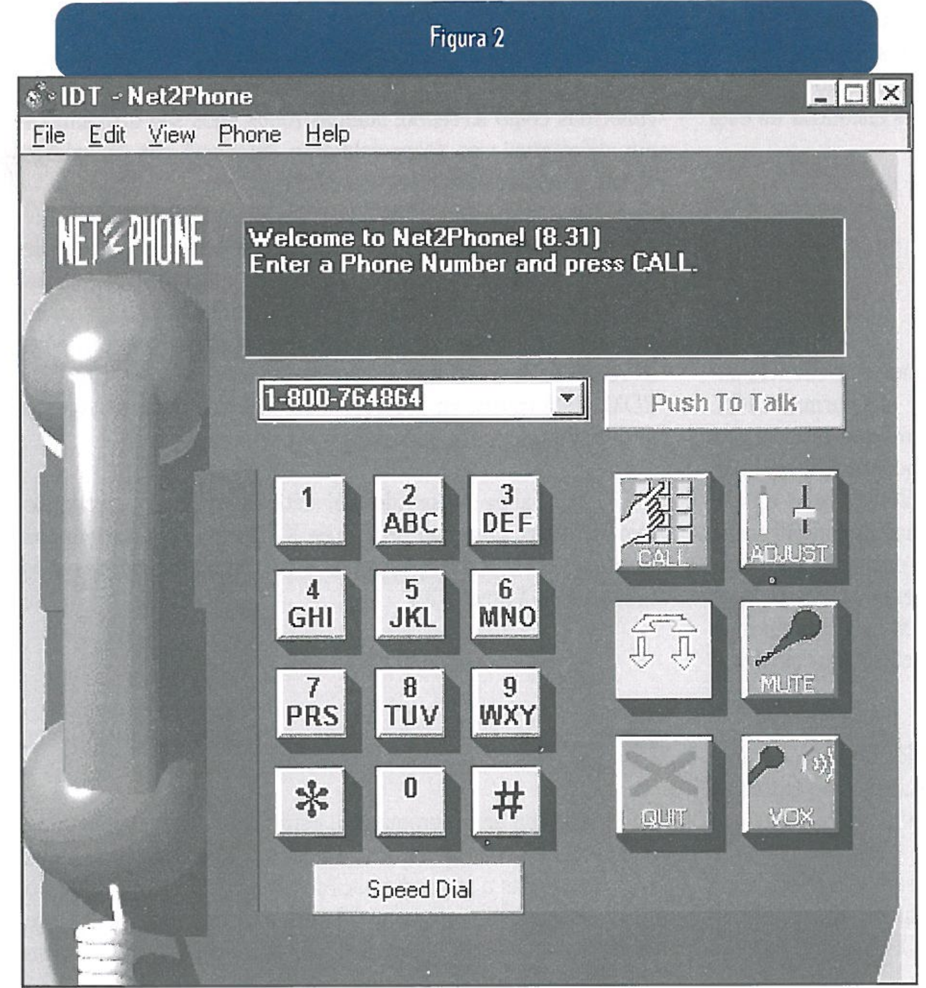


Varios

En este apartado hemos incluido una utilidad denominada 3D WinBench para medir el rendimiento hardware de nuestro PC. En su campo es de lo mejor que se puede encontrar y le recomendamos encarecidamente a todo el mundo que la ejecute a modo de prueba.

También se puede encontrar una aplicación para realizar llamadas a través del ordenador e Internet hacia teléfonos fijos de la red telefónica convencional de todo el mundo. En principio sólo soporta llamadas libres de cargo de Estados Unidos, pero también es posible darse de alta en un servidor para ampliar la zona de acción.

Figura 2



Sugerencias

Como cada mes, animamos a todos nuestros lectores a que soliciten aquellos programas que les gustaría ver incluidos en los CD-ROM de la revista, ya sea por su carácter Freeware o por su disponibilidad a modo de evaluación.

Sabemos que esto evita penosas cuentas de teléfono o esperas interminables para aquellos que están conectados a la red y facilita el acceso a recursos que de otra manera serían inaccesibles a los que no dispongan de acceso a Internet. En la medida de lo posible, iremos incluyendo los programas que nuestros lectores dessen, en los CD-ROM de próximas revistas. Nuestra dirección de correo electrónico para tales efectos es solop@towercom.es


adaptec